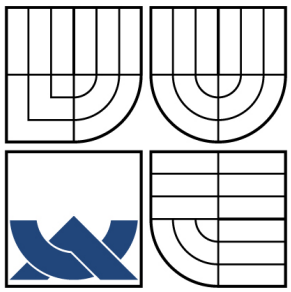


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

# NÁVRH KNIHOVNY PRO PLÁNOVÁNÍ TRAJEKTORIE ROBOTU

DESIGN OF PATH PLANNING LIBRARY FOR MOBILE ROBOT

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL NOVOTNÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. STANISLAV VĚCHET, Ph.D.

SUPERVISOR

BRNO 2008

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(Zde je vložen originál nebo kopie zadání diplomové práce)

# ANOTACE

Tato práce se zabývá analýzou problematiky plánování trajektorie ve známé mapě pomocí RRT algoritmu. Teoretická část popisuje základní pojmy a navigaci mobilního robotu. Do navigace patří lokalizace, mapování a plánování trajektorie. V každé z těchto podkapitol navigace je popsán přehled používaných metod pro lokalizaci a pro plánování cesty robotu. Praktická část popisuje implementaci navržené metody v programovacím jazyce Delphi. Jako nejlepší metoda plánování cesty robotu je zvolen algoritmus RRT stromů. Pro zajištění univerzálního komunikačního rozhraní je aplikace vytvořena jako dynamická knihovna.

# ANNOTATION

This thesis deals with analyses of problems of path planning by means Rapidly-exploring Random Trees (RRT) algorithm. The teoretic part described of basic terms and navigation mobile robots. There are localization, mapping and path planning parts of navigation. Then it is description overview of localization of methods and overview of robot path planning methods. The practical describes implementation of proposed method in Delphi. The best method for path planning of robot using RRT algorithm. For reservation universal communications interface is application creation like dynamic library.

# KLÍČOVÁ SLOVA

Plánování cesty, RRT (Rapidly-exploring Random Trees), Dynamická knihovna

# KEYWORDS

Path planning, RRT (Rapidly-exploring Random Trees), Dynamic link library

# BIBLIOGRAFICKÁ CITACE

NOVOTNÝ, M. *Návrh knihovny pro plánování trajektorie robotu.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2008. 63 s. Vedoucí diplomové práce Ing. Stanislav Věchet, Ph.D.

# PROHLÁŠENÍ

Prohlašuji, že jsem tuto diplomovou práci *„Návrh knihovny pro plánování trajektorie robotu“,* vypracoval samostatně pod vedením Ing. Stanislava Věcheta, .Ph.d. na základě použité literatury a dostupných informačních zdrojů, které jsem všechny odcitoval v seznamu literatury.

V Brně dne 23.5.2008 Podpis:

# PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Stanislavu Věchetovi, Ph.D. za odborné vedení, cenné podměty a rady potřebné při vypracování diplomové práce.

# OBSAH

# 1. ÚVOD

Plánování pohybu robotu je jednou z důležitých výzkumných oblastí robotiky. Úkolem plánování pohybu je nalezení volné cesty robotu z počáteční pozice do cílové pozice ve známé mapě se statickými překážkami. Tento problém se zdá být zdánlivě jednoduchý, ale výpočet je ve skutečnosti značně složitý.

V počátcích robotiky se studovaly především problémy, jak naplánovat cestu ve zcela známém prostředí. Až v osmdesátých letech se začaly provádět studie hledání cesty ve zcela neznámém prostředí. To byl velký krok vpřed v robotice. Bohužel vývoj robotiky začal v těchto letech mírně stagnovat a další větší rozmach mobilní robotiky se datuje až na přelomu tisíciletí. Hlavním důvodem tohoto rozvoje jsou především velmi dobré výsledky projektů, které naznačovaly, že by se mohly mobilní roboty využívat i v běžné praxi.

V posledních letech bylo vymyšleno spoustu nových plánovacích algoritmů, které jsou velmi úspěšné pro řešení náročných problémů plánování. Mezi nové plánovací algoritmy lze zařadit pravděpodobnostní plánovací algoritmy, tzv.pravděpodobnostní stromy.

Počáteční problém plánování cesty lze dále rozšiřovat o pohyblivé nebo-li dynamické překážky. Dále je možné robotu zadávat určitá omezení pohybu a také je možné vypočítat optimální trajektorii pohybu.

V současnosti je použití robotů prakticky běžnou věcí. Setkáváme se s nimi téměř ve všech oblastech lidské činnosti. Používají se buď jako praktické aplikace nebo jako předmět výzkumu. Plánování pohybu se taktéž uplatňuje i v jiných odvětvích než jen v robotice. Plánování se používá například v počítačových hrách, v počítačovém designu a také v lékařství.

V první části této práce je seznámení s problematikou navigace mobilního autonomního robotu. Navigace je rozdělena do tří částí – lokalizace, mapování a plánování trajektorie. Popisuje základní pojmy, s nimiž se lze při studiu této problematiky setkat. U každé části je ukázáno několik metod použití a následně je vybrána metoda pro realizaci praktické části.

Druhá část popisuje vybranou metodu plánování trajektorie ve známém prostředí, kterou je RRT algoritmus. Součástí metody je i její implementace v programovacím

prostředí Delphi. Aby navržený modul splňoval podmínku použitelnosti v rozdílných programovacích jazycích, je realizován jako dynamická linkovaná knihovna.

Ve třetí části jsou popsány provedené ověřovací experimenty správnosti navrženého modulu. Tyto experimenty jsou testovány v programovacím prostředích jazyka C a LabView. Poté jsou výsledky experimentu porovnávány a je provedeno zhodnocení těchto výsledků.

# 2. CÍLE PRÁCE

Prvním cílem této práce je podat přehled o používané navigaci v oblasti mobilní robotiky. Navigace je složena z lokalizace, mapování a plánování trajektorie. Cílem je popsat základní metody jednotlivých částí navigace. Poté je důležité zvážit klady a zápory jednotlivých metod a vybrat nejvhodnější metodu pro plánování trajektorie robotu.

Druhým cílem je navrhnout a realizovat zvolenou metodu plánování trajektorie. Jedním z požadavků realizace dynamického modulu je, aby modul disponoval univerzálním komunikačním rozhraním. Toho je dosaženo pomocí dynamické linkované knihovny, zkráceně DLL knihovny. Tím je dosáhnuto toho, že je možné se dostat k plánovacím funkcím nezávisle na použitém programovacím jazyce.

Posledním cílem práce je experimentální ověření výše uvedeného modulu v jiném programovacím jazyce, než ve kterém byl modul navrhnut. Je potřeba zhodnotit funkčnost a správnost řešení a porovnat řešení s původně navrženým modulem.

# 3. ROBOT A JEHO PROSTŘEDÍ

Robot, stejně jako všechny reálné objekty, se pohybuje v nějakém prostředí. Prostředí robotu je tvořeno prostorem, ve kterém se pohybuje a plní dané úkoly. Tento prostor lze rozdělit také podle funkce a možností pohybu robotu. Roboty rozdělujeme na manipulátory a mobilní roboty. Manipulátory jsou průmyslové roboty, které jsou fixovány k pevnému bodu. Manipulátor tvoří kloubová ruka, která má za úkol zajistit polohování uchopeného předmětu v prostoru. Naproti tomu mobilní roboty se mohou pohybovat ve svém prostředí. Hlavní skupinou mobilních robotů jsou kolové, pásové a kráčející roboty.

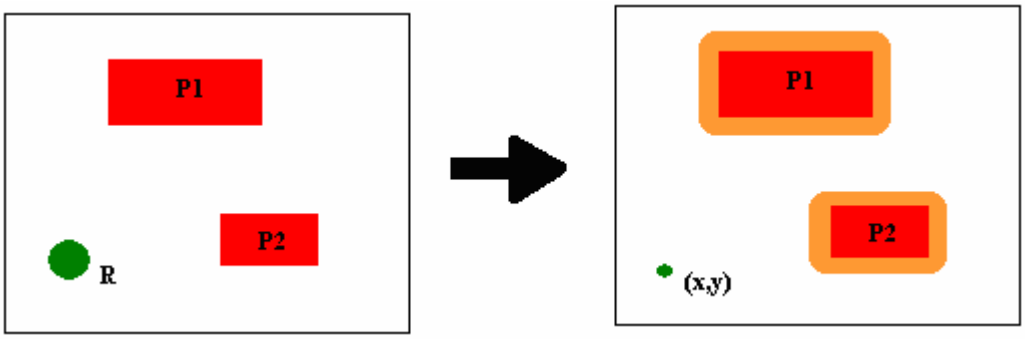
Jestliže robot dopředu zná mapu prostředí, ve kterém se bude pohybovat je potřeba zjistit, kde se nachází a poté se zjišťuje jak se přemístí do cílového místa. Tyto operace se nazývají lokalizace a plánování. Jakmile není robotu dopředu známá mapa musí si ji tvořit během cesty k cílovému místu, to znamená, že řeší zároveň lokalizaci a plánování.

## 3.1. Pracovní prostor

Pracovní prostor robotu může být reprezentován jako n-rozměrný euklidovský prostor $R^N$, kde $N=2,3$ podle toho zda jde o dvourozměrný nebo trojrozměrný prostor. V tomto pracovním prostoru se mohou vyskytovat překážky a jiná omezení robotu.

## 3.2. Konfigurační prostor *C*

Konfigurační prostor je zaveden proto, aby bylo možné matematicky popsat pozici a orientaci robotu v pracovním prostoru. Je to množina všech přípustných stavů robotu. Jeho dimenze odpovídá počtu stupňů volnosti robotu. Hlavní myšlenkou konfiguračního prostoru je reprezentace robotu jako bodu a namapování překážek do pracovního prostředí.




*Obr.1 Ukázka namapování překážek do pracovního prostoru*

Množina $C_{free}$ se nazývá volný konfigurační prostor. Tento prostor odpovídá prostoru všech konfigurací, které nekolidují s žádnou překážkou a vyhovují všem omezením robotu. Pokud jsou dodrženy stanovené podmínky nalezneme všechny přípustné pozice robotu. Opakem volného konfiguračního prostoru je $C_{obst}$. Je to prostor všech konfigurací, které kolidují s překážkou a nebo jsou porušena omezení kladená na pohyb robotu.

## 3.3. Metrika

Plánovací algoritmus potřebuje znát funkci, která popisuje vzdálenost dvou bodů (konfigurací) robotu v konfiguračním prostředí *C*. Metrika popisuje tuto vzdálenost *d*.

Metrika je zobrazení $\rho : (q_1, q_2) \in C^2 \rightarrow \rho(q_1, q_2) \geq 0$, pro které platí následující podmínky:

- Podmínka totožnosti

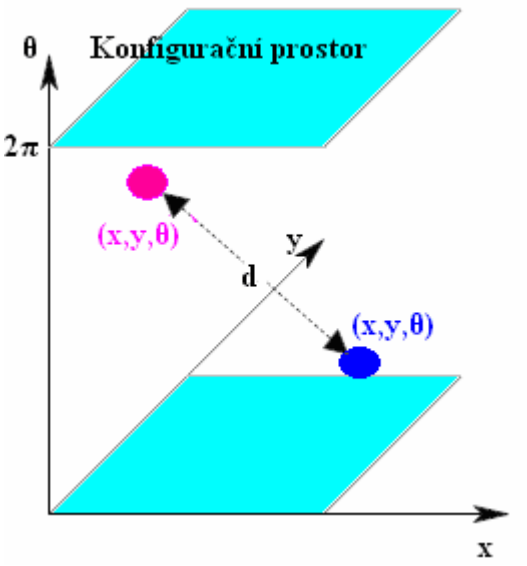
  $$\rho(q_1, q_2) = 0 \Leftrightarrow q_1 = q_2$$

- Podmínka symetrie

  $$\rho(q_1, q_2) = \rho(q_2, q_1)$$

- Podmínka trojúhelníkové nerovnosti

  $$\rho(q_1, q_2) \leq \rho(q_1, q_3) + \rho(q_3, q_2)$$




*Obr.2 Vzdálenost mezi dvěma konfiguracemi robotu*

Nejznámějším případem metriky je Euklidovská metrika. Ta vyjadřuje délku mezi dvěma body a je definována pro n-rozměrný Euklidovský prostor vztahem:

Pro $X = (x_1, x_2, ..., x_n) \in R_n$ a $Y = (y_1, y_2, ..., y_n) \in R_n$

$$\rho(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \qquad [1]$$

# 4. PROBLEMATIKA NAVIGACE ROBOTU

Automatická navigace mobilního robotu je první krok při tvorbě jeho umělé inteligence. Navigaci můžeme rozdělit do tří fází:

- Lokalizace
- Mapování
- Plánování trajektorie

Pro zajištění spolehlivé navigace je vhodné, aby robot znal svou pozici. Proto je odhad pozice na základě informací ze senzorů jedním z hlavních problémů mobilní robotiky. Tento problém se nazývá lokalizace. Jakmile jsou známé souřadnice robotu mohou se provádět další úkoly jako je tvorba map a plánování trajektorie. Tvorba mapy je realizována jako vnitřní model prostředí, který robot používá k plánování a lokalizaci. Plánování trajektorie je část, kdy robot již zná svou pozici a také mapu prostředí, ve kterém bude hledat cestu. Orientaci ve známém prostředí lze převést na problém nalezení cesty z místa A do místa B. Robot vykonávající takovou cestu se na místo určení může dostat několika různými způsoby. Existuje několik různých algoritmů, které popisují plánování trajektorie robotu.

# 5. LOKALIZACE

Lokalizace je schopnost mobilních robotů určit svou pozici. Například člověk s turistickou mapou, který se chce dostat na konkrétní místo. Nejprve musí zjistit, na kterém místě se nachází, aby vůbec mohl použít mapu. To provede pomocí významného objektu na mapě (ulice, budova, hora atd.). Nejlepšími objekty jsou takové objekty, které se na mapě vyskytují jen jednou. Může se také stát, že z jednoho místa nelze určit polohu, protože není v blízkosti žádný objekt podle, kterého by bylo zřejmé, kde se turista nachází. Je tedy nutné během pohybu sbírat informace o okolí a snažit se najít místo, kde bude možné určit pozici. Tato schopnost mobilních robotů se nazývá autolokalizace nebo zkráceně lokalizace.[3] Tento problém může být dále rozdělen na tři části:

- Sledování pozice
- Lokální lokalizace (počáteční pozice robotu známá)
- Globální lokalizace (počáteční pozice robotu neznámá)

Lokalizace mobilního robotu ve známé nebo v neznámé mapě se provádí určitými metodami. Mezi nejznámější metody patří:

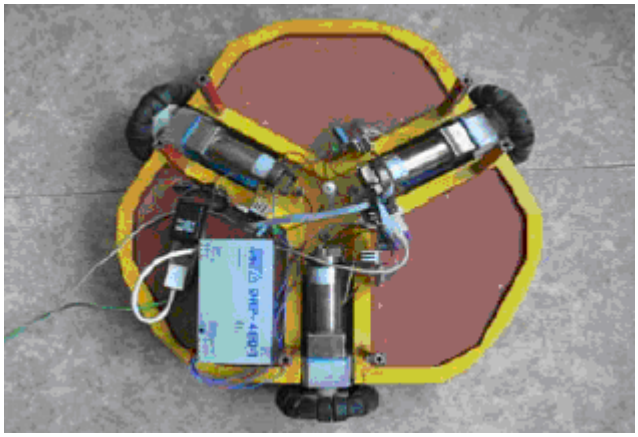
- Dead reckoning
- Lokalizace využívající aktivních prvků
- Markovská lokalizace
- Lokalizace Monte Carlo
- Metoda PCSM (Pre-Computer Scan Matching)

## 5.1. Dead reckoning

Tato metoda je založena na základě informací o změnách vnitřního stavu robotu. Je to matematická procedura pro určování současné pozice robotu pomocí postupného přičítání díky známému kurzu a rychlosti v průběhu času. Nejznámější metodou dead reckoning je odometrie.

Odometrie je nejrozšířenější metoda pro určování polohy robotu. Tato metoda se zabývá odhadem aktuální pozice a určením vzdálenosti pohyblivého objektu. Aktuální pozice se určuje v závislosti na počáteční poloze, ve které se pohyblivý objekt nacházel, než došlo k jeho přesunutí. Určení vzdálenosti z počáteční pozice k aktuální pozici se u

kolových mobilních robotů nejčastěji realizuje pomocí inkrementálních snímačů. Principem inkrementálního snímače je takový, že otáčky kol se převádí na lineární posuv vzhledem k podložce. To vede k akumulaci chyb. Podmínkou přesné odometrie je tedy znalost geometrického modelu vozidla. Všechny typy podvozku nejsou vhodné pro aplikaci odometrie. Příkladem vhodného podvozku je diferenciální, trojkolový a všesměrový podvozek. Na obr.3 je uveden všesměrový robot OMR III, který byl navržen a realizován na fakultě VUT FSI BRNO.



*Obr.3 Všesměrový robot OMR III*

Metody založené na dead reckoning jsou zatíženy kumulativní chybou. Malé chyby v senzorických datech se často sčítají až dosáhnou takové hodnoty, že je výsledná poloha nepoužitelná. Přesto se tato lokalizace dost často používá, ale musí být jako součást složitější lokalizace. Informace jsou potom skládány s dalšími informacemi od jiných senzorů.[8]

## 5.2. Lokalizace využívající aktivních prvků

Umělé objekty jsou pro navigaci robotu mnohem vhodnější než přirozené objekty, protože jsou vytvořené speciálně se záměrem, aby byly co nejsnáze detekovatelné. Je potřeba je speciálně nainstalovat do prostředí, ve kterém se bude robot pohybovat. Jako umělé objekty se používají například barevné značky na zdi, majáky vysílací ultrazvukové nebo světelné signály apod. Polohu robotu určíme v závislosti na rozmístění jednotlivých majáků, a proto není třeba mít mapu prostředí.

Nevýhodou je, že omezují pracovní prostor robotu pouze na oblasti, v nichž se nacházejí. Navíc je nutný zásah do vnějšího prostředí, který nemusí být v některých

případech možný. Nejznámější takovou lokalizací je lokalizace prostřednictvím GPS, která využívá družic rozmístěných na oběžné dráze.

## 5.3. Markova lokalizace

Tato lokalizace je založena na pravděpodobnostním přístupu. Metoda je schopna lokalizovat robot i ve stavu, kdy je jeho startovní pozice neznámá. Slouží jako základ pro metodu Monte Carlo.[3]

Podstatou markové lokalizace je určení pravděpodobnosti s jakou se robot nachází na dané pozici. Pravděpodobnost $Bel(L_T = l)$, kde $l$ je pozice robotu v prostoru, kde $x$, $y$ jsou kartézské souřadnice a α je orientace robotu. Pokud není známa počáteční pozice robotu, je hustota pravděpodobnosti $Bel(L_0)$ rovnoměrně rozdělená. Rozdělení pravděpodobnosti je aktualizované po každém získání nových dat ze senzorů.

Když se robot pohybuje, je získané rozložení pravděpodobnosti polohy robotu aktualizováno jako

$$Bel(L_T = l) = \sum_{l'} p_a(l / l')Bel(L_{T-1} = l'), \qquad [2]$$

kde $p(l /l')$ je pravděpodobnost, se kterou se robot přemístil z pozice $l$ do pozice $l'$ pomocí akce $a$. Předpokládáme, že chyby v poloze i orientaci mají gaussovské rozdělení se střední hodnotou nula a jsou úměrné délce pohybu.

Po tom, co robot získá data ze senzorů snímajících okolí $s_T$ provede aktualizaci pravděpodobnosti podle vztahu:

$$Bel(L_T = l) = \alpha_T p(s_T / l)Bel(L_{T-1} = l), \qquad [3]$$

kde $p(s_T /l)$ je pravděpodobnost, že senzorová data jsou získána v pozici $l$ a $\alpha_T$ je normalizační koeficient, který za důsledek následující podmínku:

$$\sum_{l} Bel(l) = 1 \qquad [4]$$

## 5.4. Lokalizace Monte Carlo

Tato metoda vychází z Markovy lokalizace. I zde je charakteristickou vlastností způsob reprezentace hustoty pravděpodobnosti popisující odhad pozice robotu. Využívá se množiny vzorků, které jsou ohodnoceny váhami. Rozložení ve stavovém prostoru určuje, jak je která pozice pravděpodobná.[9]

$$S = \{s_i / i = 1,...,n\},$$
$$s = \{1, p\}, l = \{x, y, \alpha\}, \qquad [5]$$

kde $n$ je počet vzorků , $x$ a $y$ je poloha robotu, α je orientace a $p$ je pravděpodobnost pozice, pro kterou platí tyto podmínky:

$$p \geq 0, \sum_{i=1}^{n} p_i = 1 \qquad [6]$$

Počáteční hodnotu pravděpodobnosti $p$ volíme:

$$p_i = \frac{1}{n} \qquad [7]$$

Z čehož plyne, že hustota pravděpodobnosti je rovnoměrně rozložená.

Algoritmus má dvě základní části:[8]

- Predikce – přesun celé množiny vzorků na základě informací o změně pozice robotu např.z odometrie
- Korekce – korekce vah jednotlivých vzorků množiny je založena na shodě či neshodě naměřených dat s předpokládanými daty, která by měla odpovídat pozici reprezentované příslušným vzorkem

Po provedení predikce je získáná nová množina vzorků, která reprezentuje upravenou hustotu pravděpodobnosti na základě informací o změně pozice.

Korekce spočívá v úpravě vah jednotlivých vzorků množiny. Každý vzorek je ohodnocen vahou na základě získaného měření dat ze senzorů. Toto měření dat odpovídá předpokladu pro příslušnou pozici hodnoceného vzorku.

Při opakování korekčního kroku dojde k znehodnocení množiny vzorků. Většina vzorků množiny bude mít zanedbatelné váhy a pár vzorků bude mít váhy obrovské. Takové

rozložení vah není optimální. Pravděpodobnost výskytu robotu na pozici, jejichž vzorky mají malou váhu je minimální, a proto je zbytečné se jimi zabývat. Naproti tomu pravděpodobnost výskytu robotu na pozici, kde jsou vzorky s vysokou vahou je více než pravděpodobné. Proto se často využívá ještě fáze nazvaná jako převzorkování. Úkolem převzorkování je vyloučit vzorky s hodně malou vahou a vzorky s velkou vahou naopak rozdělit na několik vzorků.

## 5.5. Metoda PCSM (Pre-Computed Scan Matching)

Princip metody PCSM (Pre-Computed Scan Matching – porovnání přepočítaných scanů) je na základě porovnání předem pořízeného snímku celého okolí s aktuálním snímkem okolí. Hlavní úkolem je odhadnout pravděpodobnost hustoty okolních prázdných dat.[3]

Typickým příkladem je:

$$Bel(x) = r(x, a, S), \qquad [8]$$

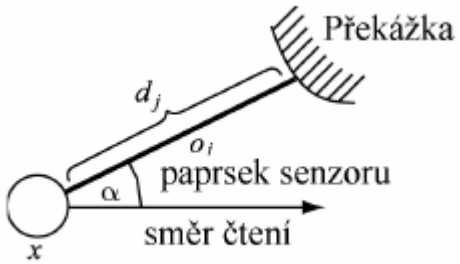
kde $x$ je pozice robotu, $a$ je označení aktuálního snímku okolí robotu v daném stavu, $S$ reprezentuje soubor m vzorků rozdělených rovnoměrně v prostoru, $r$ je funkce, která vrací ohodnocení pro existující vstupy $x$, $a$, $S$.

Ohodnocení jednotlivých vzorků je dáno:

- Získání vzdálenosti od překážky

$$d_j = g(x, o_j), \qquad [9]$$

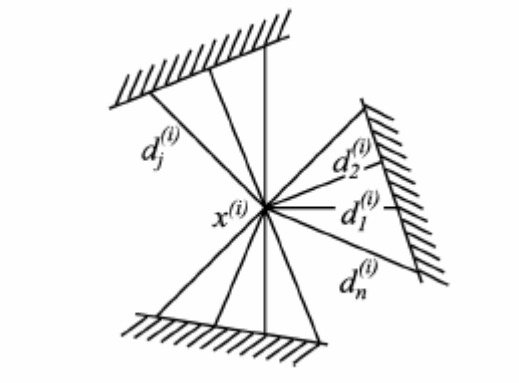
kde $x$ je předpokládaná pozice robotu, $o_j$ ideální naměřená vzdálenost, $g(x, o_j)$ měření ideálním senzorem, $d_j$ je výsledná vzdálenost od překážky.




*Obr.4 Jednotlivá měření vzdálenosti*

- Z n paprsků získáme scan

$$d = (d_j)_{j=1,\dots,n} \qquad [10]$$




*Obr.5 Kompletní scan*

- Sada S z m vzorků

$$S = (x^{(i)}, d^{(i)})_{i=1,\dots,m} \qquad [11]$$

- Výsledná funkce r je:

$$r(x^{(i)}, a, S) = \sum_{j=1}^{n} (d_j^{(i)} - a_i)^2 \qquad [12]$$

# 6. MAPOVÁNÍ

Mapování prostředí je důležitou schopností mobilního robotu. Mapa je modelem prostředí, ve kterém se robot pohybuje. Můžeme ji rozdělit na dvě části. První je mapa, která je robotu již předem známa. U druhé metody si mapu prostředí musí robot autonomně budovat během pohybu prostředím, to znamená, že pohybující se mobilní robot má za úkol zaznamenávat cestu, kterou již prošel a tím si vytvářet mapu prostředí. Aby byla mapa použitelná pro naplánování cesty, měla by vhodně reprezentovat minimální znalost o volném prostoru.

Druhá možnost vytváření mapy je mnohem vhodnější, poněvadž předem známé mapy prostředí nebývají vždy k dispozici a také nemusí odpovídat aktuálnímu stavu prostředí.[2]

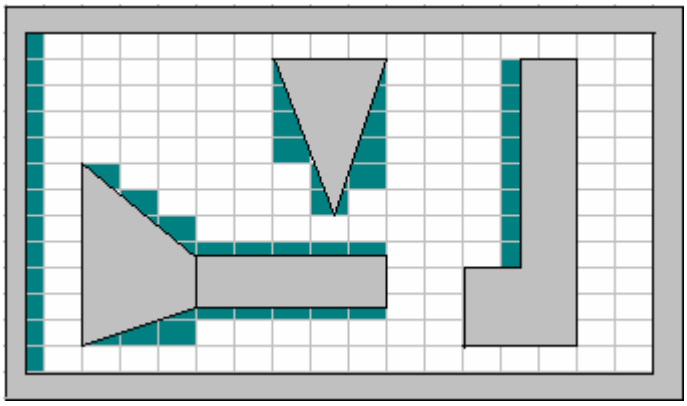
Mapy rozdělujeme na 3 skupiny:

- Senzorická
- Geometrická
- Topologická

## 6.1. Senzorická mapa

Senzorická mapa je nejnižší znázornění prostředí. Princip spočívá ve vhodně uspořádaných a uchovaných datech. Jednoduchým příkladem senzorické mapy je mřížka obsazenosti. Prostředí je rozděleno na stejné dílky uspořádané do mřížky. Každý dílek mřížky, někdy také nazývána jako buňka mřížky, v sobě uchovává pravděpodobnost o tom, zda je prostor volný nebo je obsazený překážkou. Senzorická mapa má výhodu jednoduché tvorby a možnosti přidávání dat přímo při tvorbě mapy. Nevýhodou je paměťová náročnost reprezentace.

Na obr.6. jsou šedou barvou vyznačeny překážky. Jelikož se za překážku považují i ty buňky, jejíž prostor překážka vyplňuje jen částečně tak jsou tyto prostory vyznačeny modře.



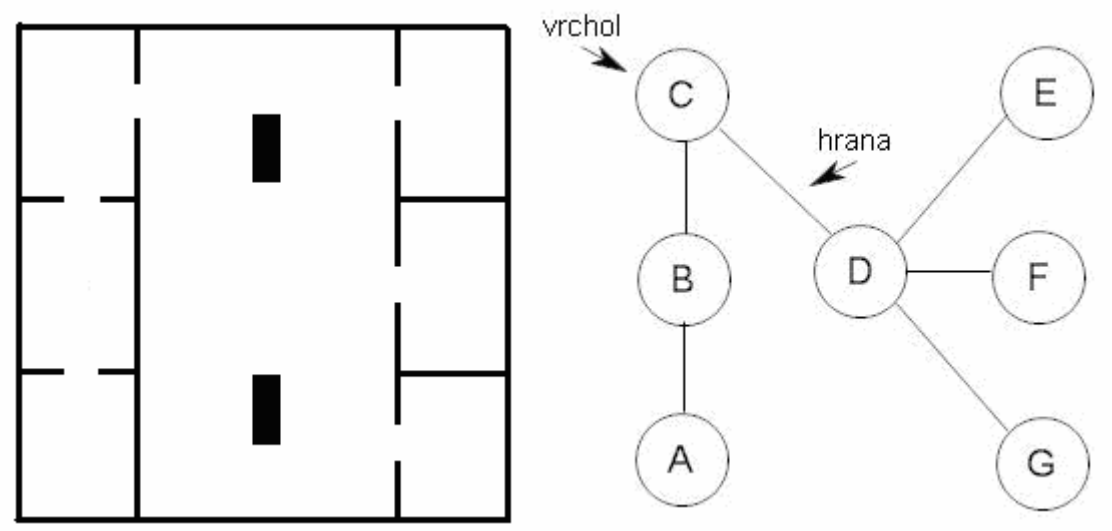
*Obr.6 Senzorická mapa*

## 6.2. Geometrická mapa

Geometrická mapa znázorňuje prostředí geometrickými prvky, jako jsou přímky, kružnice, čtverce, polygony apod. Tato mapa je více či méně abstraktní mapou prostředí, jelikož abstrahuje model od přílišných detailů. Tvorba mapy je již výpočetně náročnější, protože se musí najít jednoduchý geometrický prvek, který nejvěrohodněji popíše a namodeluje naměřená data. Tato mapa je mnohem příznivější pro plánování trajektorie. Taktéž je tato mapa výhodnější z hlediska paměťové náročnosti a to díky zavedené abstrakci.

## 6.3. Topologická mapa

Nejabstraktnější modelem prostředí je topologická mapa. Prostředí je modelováno grafem. Graf je reprezentován uzly a hrany. Uzly znázorňují jednotlivá místa, která jsou pro robot jakýmkoli způsobem důležitá. Hrany uzlu reprezentují přemístění robotu z jednoho místa do druhého. Topologické mapy nemusí obsahovat žádnou geometrickou informaci.




*Obr.7 Prostředí a topologická mapa*

# 7. PLÁNOVÁNÍ TRAJEKTORIE VE ZNÁMÉ MAPĚ

Plánování se skládá z globálního a lokálního plánovače. Globální plánování se provede ještě před uvedením robota do pohybu a vyžaduje, aby byla kompletně známa mapa prostředí, a to včetně statických překážek. Vlastní pohyb je poté řízen lokálním plánovačem.

Důležitou částí inteligentního systému je to, že si dokáže vytvořit vnitřní model prostředí. Jakmile je zadán startovní a cílový stav je možné nalézt takovou posloupnost akcí, aby při jejich vykonání došlo k přemístění ze startovního do cílového stavu. Jako plán se potom bere tato posloupnost akcí. Stavový prostor, ve kterém se robot pohybuje může být převeden na orientovaný graf, kde uzly jsou jednotlivé stavy a hrany jsou přechody mezi stavy. Uzly a hrany jsou ohodnoceny funkcemi, které vypočítají jejich cenu vzhledem k různým okolnostem.

K nalezení vhodné cesty z místa A do místa B se použije některá z plánovacích metod. Robot vykonávající takovou cestu se na místo určení může dostat několika různými způsoby. V nejčastějších případech se splňují následující cíle:

- Jak se dostat z bodu A do B
- Jak se vyhnout překážkám na cestě
- Jak najít nejkratší možnou cestu
- Jak najít příslušnou cestu rychle

Existují algoritmy, které splňují některé nebo i všechny tyto body, nebo naopak nesplňují žádný z uvedených bodů. V následujícím přehledu je uvedeno několik metod pro plánování cesty ve známé mapě.

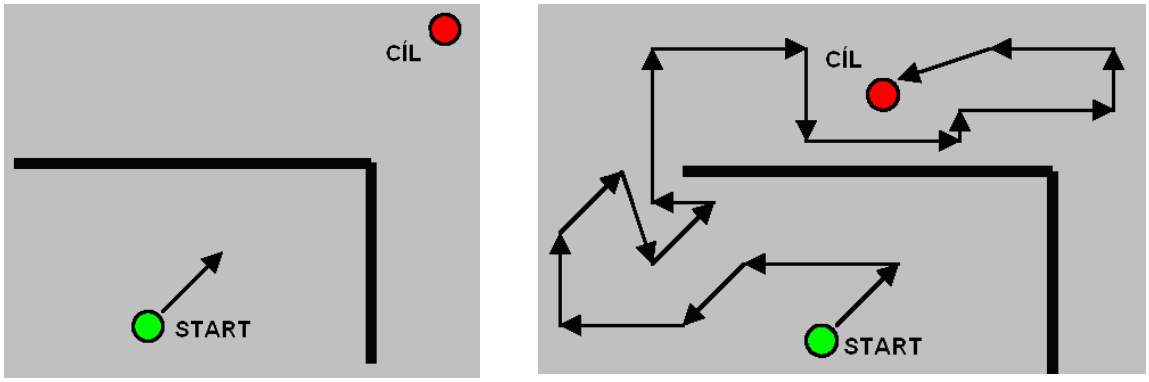
## 7.1. Neinteligentní algoritmy

Za neinteligentní algoritmus je považován nejjednodušší algoritmus. Tento algoritmus má výhodu v tom, že nemusí znát podrobně své okolí. Pro pohyb robotu v okolí stačí jen jednoduché informace, které mohou být typu „dál nemůžu, přede mnou je stěna“.

Jako příklad nejjednoduššího algoritmu je považován algoritmus, který vypočítává směr k cíli a po této přímce se robot vydá. Tento algoritmus nefunguje na členitém terénu. Jakmile se robot dostane ke stěně zarazí se a nemůže pokračovat dál. Tento algoritmus je ve většině případu nevhodný.

V každém případě je tento algoritmus základní, protože ostatní algoritmy mohou rozdělit cestu na množinu bodů, mezi kterými nejsou žádné překážky a robot pak vykonává cestu mezi těmito body.

Další algoritmus je založený na myšlence náhodného pohybu. Tento algoritmus se dá použít i v případě, že robot nezná prostor, ve kterém se pohybuje. Navíc lze rozšířit o řadu heuristik, které zvýší pravděpodobnost a rychlost nalezení cesty.




*Obr.8 Nalevo je algoritmus přímo k cíli, napravo je algoritmus náhodného pohybu*

## 7.2. Metody rozkladu do buněk

Tato metoda nejdříve rozdělí konfigurační prostor $C_{free}$ do jednotlivých buněk. Tyto buňky mohou být libovolného tvaru, ale pro jednoduchost se volí čtvercové nebo-li mřížkové rozdělení konfiguračního prostoru. Buňky se rozdělí na volné a obsazené podle toho, jestli obsahují překážku či nikoli. Z volných buněk se vytvoří vrcholy grafu. Graf je poté tvořen vrcholy reprezentující volné buňky a hrany mezi vrcholy. Cesta grafem ze startovního vrcholu do cílového vrcholu je vyhledána některým z daných algoritmů.

### 7.2.1. Plánování na mřížce

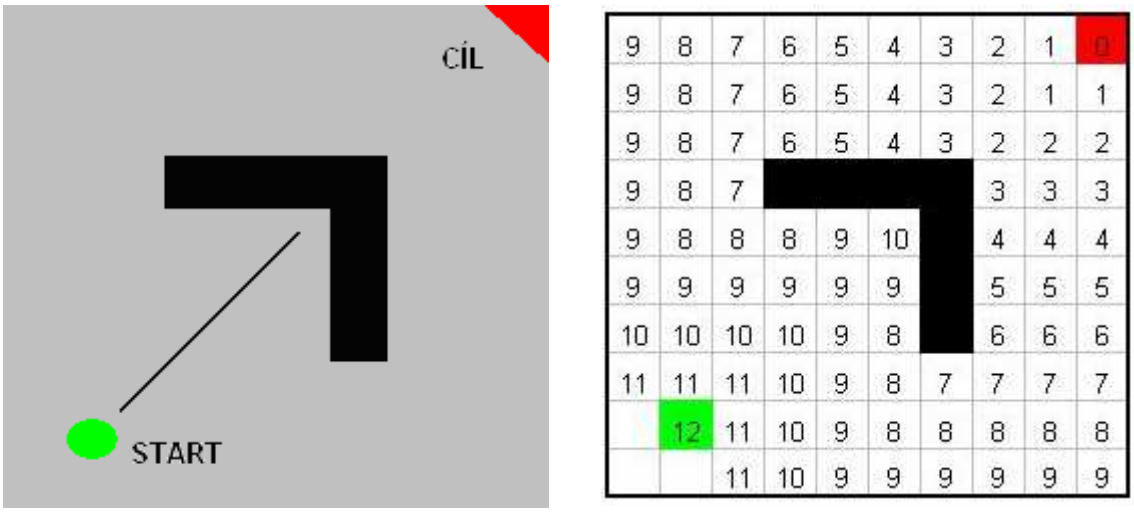
Algoritmus plánování na mřížce je založen na senzorické mapě, která rozdělí celý prostor na oblasti obsazené překážkou a na volný prostor. Složitost plánovacího algoritmů je dána počtem a detailnosti popisu překážek v prostředí. S výhodou je možno použít i hierarchické reprezentace prostředí. U tohoto prostředí se nejdříve hledá cesta v úrovní nejvyšší, která má nejmenší rozlišení a její prohledání je tedy nejrychlejší. Pokud není cesta nalezena sestoupí se vždy o úroveň níže.

Vylepšení pro plánování na mřížce je realizováno potenciálovými poli. Vylepšení spočívá v tom, že každé buňce v mapě je přiřazena hodnota. Tato hodnota udává směr, kterým směrem se má robot pohybovat. Cíli je přiřazena nejnižší hodnota a startu naopak nejvyšší. Při pohybu takto ohodnocenou mřížkou se postupuje od nejvyšší hodnoty k hodnotě nejnižší. Vyhnutí se překážky v mapě je ošetřeno tak, že se překážkám přiřadí hodnota vyšší než kdekoli v mapě.

Tento algoritmus funguje pouze v případě, že prostředí obsahuje pouze konvexní překážky. Za použití překážek jiného tvaru hrozí problém uvíznutí v místě, kde potenciálová funkce tvoří lokální minimum. Lokální minimum je tvořeno buňkou, která není cílem a buňky vedle ní jsou buď překážky nebo mají vyšší hodnotu potenciálu.

Na obr.9 je realizován vylepšený algoritmus. Princip vytváření tohoto algoritmu je takový, že se začne v cíli, kterému je přiřazena nulová hodnota. Od cíle se pokračuje tak, že se vezmou všechny neočíslované buňky sousedící s již očíslovanou buňkou a přiřadí se jim hodnota o jedno vyšší. Tento postup se opakuje do té doby než je očíslovaná startovní pozice nebo dokud nejsou nevyčerpány všechny neočíslované buňky. Aby nedošlo k uvíznutí v lokálním minimu je této buňce přiřazena co nejvyšší hodnota. Tento

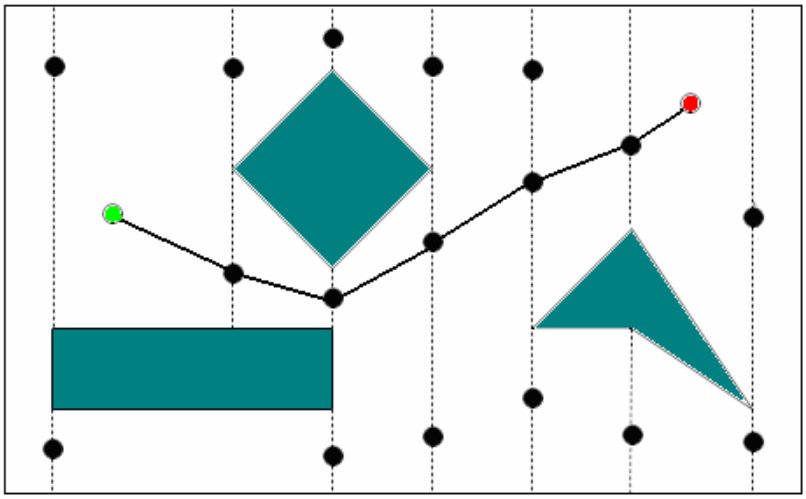
algoritmus může určit všechny buňky, ze kterých se lze dostat od startovní pozice až k cíli. Číslo přiřazené dané buňce udává vzdálenost od cíle.[8]




*Obr.9 Nalevo je lokální minimum a napravo je vylepšený algoritmus*

### 7.2.2. Exaktní rozklad

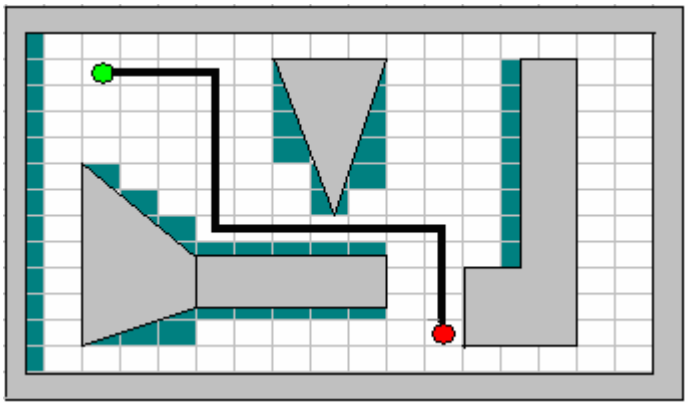
Exaktní rozklad může mít více variant. Záleží na tvaru použitých buněk. Jejich tvar se volí jednoduchý tak, aby bylo možné snadno spočítat cestu mezi dvěma pozicemi robotu. Proto se buňky často volí jako lichoběžníkové nebo trojúhelníkové. Tato metoda nalezne vždy cestu nebo ověří, že cesta neexistuje. Exaktní rozklad je tedy kompletní algoritmus.



*Obr.10 Nalezená cesta pomocí exaktního rozkladu*

### 7.2.3. Aproximativní rozklad

Tato metoda patří do skupiny plánování na mřížce, protože se celý pracovní prostor (i překážky) rozdělí do buněk a vznikne mřížka. Všechny buňky mřížky jsou stejného tvaru. Nejjednodušší je tvar čtverce, ale je možné rozložit pracovní prostor do kvadrantů, u nichž není velikost buňky konstantní. U této metody může dojít k tomu, že není cesta nalezena, přičemž ve skutečnosti cesta existuje. Je to zapříčiněno tím, že buňka, jejíž prostor překážka vyplňuje jen částečně je považována za buňku obsazenou překážkou.



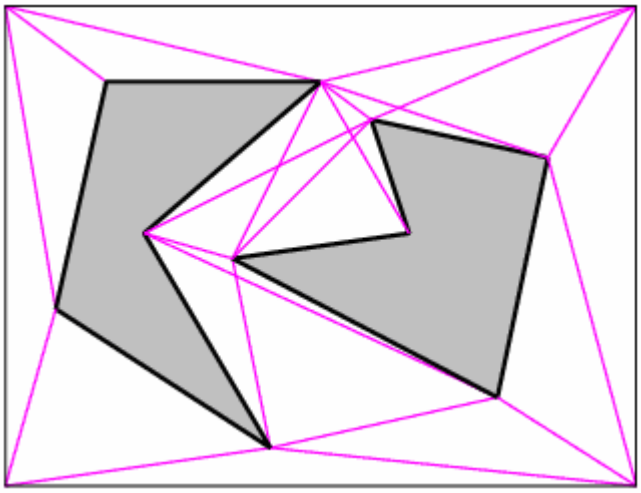
*Obr.11 Aproximativní rozklad na čtvercové buňky*

## 7.3. Mapy cest

Do této skupiny plánovacích metod patří algoritmy, které vytváří graf reprezentující volný prostor. Orientovaný graf tvoří vrcholy a hrany. Vrcholy grafu tvoří křižovatky a hrany cesty, po kterých se může robot pohybovat. Jakmile je do grafu přidán startovní a cílový vrchol dochází k řešení problému hledání cesty v grafu. K nalezení cesty se používají algoritmy pro hledání cesty v grafu a některé jsou popsány níže.

### 7.3.1. Graf viditelnosti

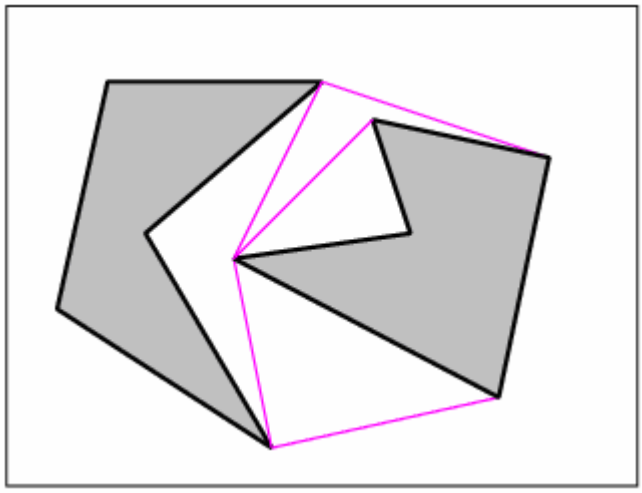
Graf viditelnosti slouží k popisu známého prostředí. Vrcholy grafu tvoří vrcholy překážek a také startovní a cílový vrchol. Hrany grafu představují viditelné spojení vrcholů, které neprocházejí žádnou překážkou. Cesta volným prostorem je pak určena posloupností hran v grafu. Tuto metodu je možné použít jen za předpokladu, že je prostor tvořen polygonálními překážkami. Jestliže je prostor tvořen nepolygonální překážkou tak se musí nahradit polygonem, který co nejlépe popisuje danou překážku. U tohoto typu grafu však dochází k značnému nárůstu vrcholů a díky tomu se prodlužuje doba k nalezení cesty grafem. Obr.12 je přejat z [8].



*Obr.12 Graf viditelnosti*

### 7.3.2. Redukovaný graf viditelnosti nebo-li graf tečen

Graf tečen řeší výrazný nárůst vrcholů u grafu viditelnosti. Princip spočívá v tom, že hrany jsou zredukovány pouze na tečny. Hledání cesty je díky menšímu počtu hran rychlejší a je možné tuto metodu plánování cesty použít i na nepolygonální překážky. Obr.13 je přejat z [8].



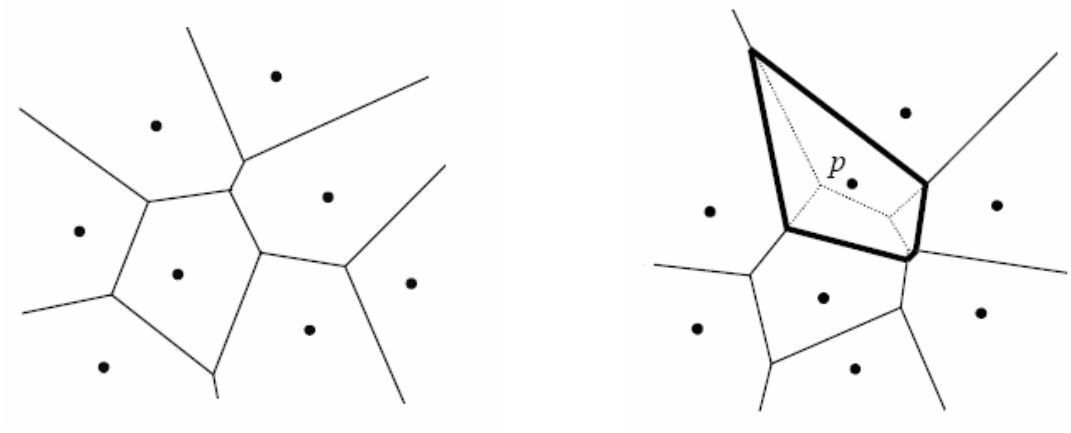
*Obr.13 Redukovaný graf viditelnosti*

### 7.3.3. Algoritmy pro konstrukci Voronoiva diagramu

- Inkrementální algoritmus
- Algoritmus rozděl a panuj

Inkrementální algoritmus vytváří diagram postupným vkládáním bodů. Výchozím stavem je diagram se dvěma body z dané množiny bodů. Tyto body jsou od sebe odděleny kolmicí, která prochází středem jejich spojnice. K stávajícímu diagramu se přidá jeden bod, který je označen bodem *p*. Algoritmus nejprve určí aktuální pozici bodu *p*. Tato aktuální pozice je určení kam bod *p* patří. Bod, který definuje oblast, kde se nachází bod *p* je nazvána *q*. Kolmice vedená středem úsečky *pq* vytvoří novou hranu *h*. Koncové body hrany *h* jsou průsečíky $x_1$, $x_2$ úsečky *pq* s hranami oblasti obsahující bod *p*. Poté se vede kolmice jedním z těchto průsečíků na spojnici bodů definujících Voronoiovy oblasti. Tato kolmice se protne s další hranou v průsečíku $x_3$. Pokračováním tohoto postupu se dojde až do bodu $x_k = x_1$. Nová Voronoiova oblast vznikne posloupností hran procházejících body $x_1$, $x_2$ „…, $x_1$. Poslední fáze je odstranění všech úseků hran obsažených uvnitř nově vytvořené oblasti. Tento způsob vkládání bodů vyžaduje čas $O(n^2)$.

Jestliže je použit náhodný způsob vkládání nových bodů většinou se inkrementální algoritmus zrychlí. Tento algoritmus bude poté vyžadovat čas jen $O(n \log n)$.

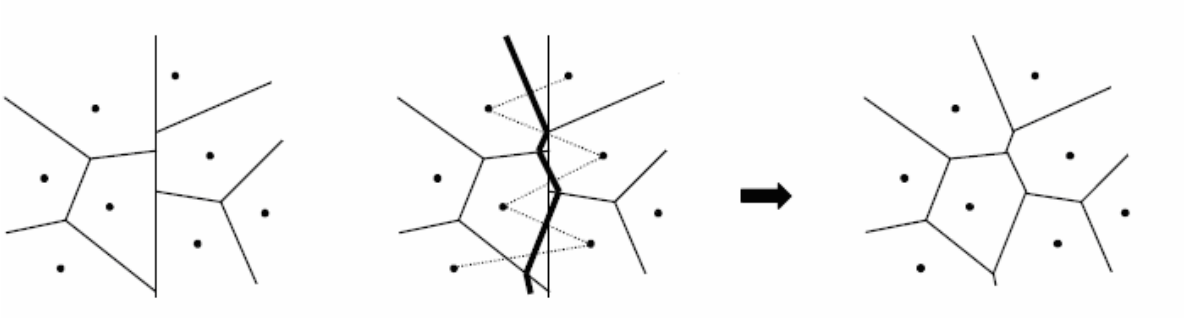



*Obr.14 Nalevo Voronoiův digram, napravo Inkrementální algoritmus*

Algoritmus rozděl a panuj rozdělí danou množinu n-bodů svislou čarou na dvě podmnožiny $S_1$ a $S_2$ podobné velikosti.

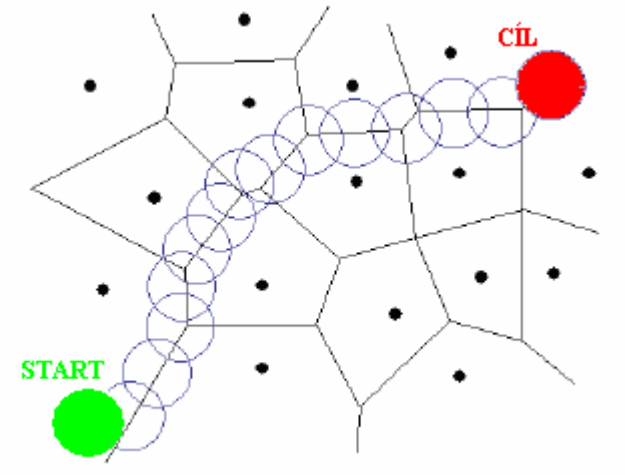
Princip spojení dvou diagramů v jeden je založen na vyhýbání se nebezpečným objektům, které jsou reprezentovány danými body. Aby se minimalizovalo nebezpečí

postupuje se po cestě vedoucí kolmo na střed úsečky spojující nejbližší dva body. Tím je zaručena největší vzdálenost od obou bodů. Kdykoliv dojde k protnutí s hranou, jejíž koncové vrcholy leží v $S_1$, dojde ke změně směru od $S_1$ bodu. Obdobně to platí i pro protnutí hrany v $S_2$.



*Obr.15 Spojení dvou dílčích diagramů v jeden*

V uvedeném příkladu je ukázáno využití daného algoritmu v robotice. Algoritmus má za úkol najít cestu robota ze startovní pozice do cílové pozice. Z překážek se stanou řídící vrcholy a ty rozdělí prostor na *n*-oblastí. Jednotlivým oblastem je přiřazen jen jeden řídící bod a platí, že všechny body z oblasti mají nejblíže ke svému řídícímu bodu. Voronoi hrany udávají hranice oblastí. Voronoi hrany mají stejnou vzdálenost ke dvěma nejbližším překážkám. Proto aby nedošlo ke kolizím robota s některou z překážek musí se robot pohybovat po hranách Voronoiova diagramu. Konečný algoritmus je takový, že se robot nejdříve posune ze startovní pozice na Voronoi hranu a pak hledá cestu v grafu.[4]




*Obr.16 Plánování cesty robotu ve Voronoi diagramu*

### 7.3.4. BSP stromy

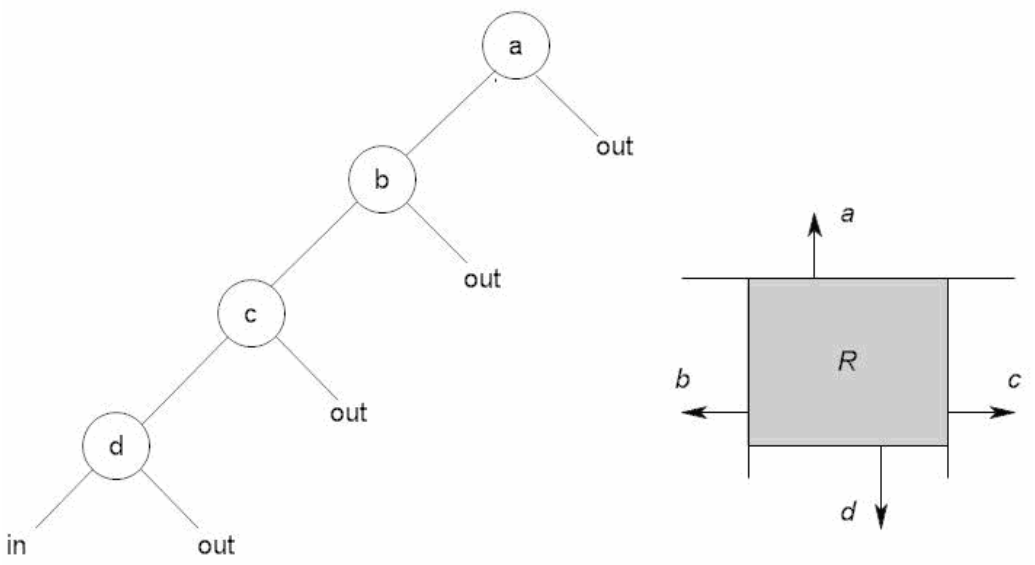
BSP stromy je technika správy polygonů v prostoru tak, aby umožnila jejich snadné vykreslování. Umožňuje také snadno detekovat kolize a automaticky tvořit portály. BSP také splňuje podmínky binárního vyhledávacího stromu.

Libovolný mnohostěn je možné popsat jako oblast ohraničenou určitým počtem nadrovin. Tyto nadroviny je možné uspořádat do BSP stromu, kde každý uzel představuje jednu nadrovinu, listy stromu jsou vnitřní oblasti mnohostěnu nebo prázdný prostor.

V libovolném BSP stromu jsou obsaženy vnitřní uzly, prázdné listy a plné listy. Vnitřní uzel je uzel BSP stromu, který má dva následníky a obsahuje rovnici dělící nadroviny. Každý vnitřní uzel je vlastně kořenovým uzlem příslušného podstromu. Listy jsou takové uzly stromu, které nemají žádného následníka a neobsahují ani žádnou geometrickou informaci. List stromu vždy odpovídá nějaké konvexní množině bodu v prostoru. Dále už jen nese informaci o tom, zda je tento list plný nebo prázdný. Plný list je takový list, který představuje vnitřní část mnohostěnu. Prázdný list představuje prázdný prostor.

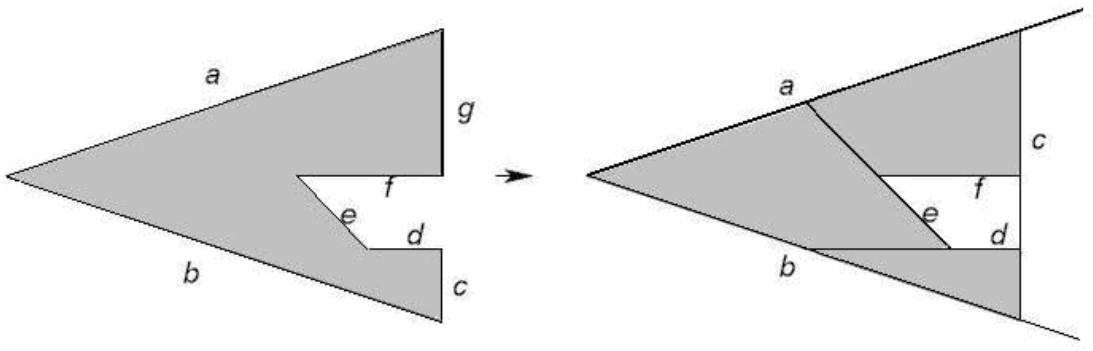
BSP strom konvexního mnohostěnu vytvoříme tak, že nejprve zavedeme kořenový uzel, který nemá žádné následníky. Vybereme jednu nadrovinu popisující konvexní mnohostěn a umístíme ji do kořenového uzlu stromu. Poté vytvoříme dva následníky tohoto uzlu, které označíme jako front a back. Vše, co se nachází v podstromu, jehož kořenem je uzel *front*, patří kladnému podprostoru. Tento uzel můžeme označit jako vnitřní oblast mnohostěnu. Následující back označíme jako vnější oblast mnohostěnu a pokračujeme uzlem front. Tomuto uzlu přiřadíme další z nadrovin ohraničujících mnohostěn. Poté stejným způsobem vytvoříme následníky. Celý postup opakujeme pro všechny nadroviny mnohostěnu. Výsledkem je velmi jednoduchý BSP strom, jehož back uzly jsou listy označené jako prázdné (*out*) a všechny front uzly obsahující dělící nadroviny. Teprve poslední uzel front je plným listem označeným jako vnitřní oblast (*in*) mnohostěnu.

Libovolný konvexní mnohostěn ohraničený *n* nadrovinami potom vytváří strom s *n* vnitřními uzly, n prázdnými listy a jedním plným listem. Na obr.17 je zachycen čtverec, který je popsán čtyřmi nadrovinami *a,b,c,d,* které tvoří uzly stromu. Vnější oblast čtverce představují čtyři listy, označené jako *out* a vnitřní oblast čtverce je jediným *in* listem.




*Obr.17 Konvexní polygon (čtverec) a jeho BSP strom*

Konstrukce BSP stromu nekonvexních mnohostěnů je popsána jako sjednocení několika menších konvexních mnohostěnů. Vznikne složitější strom, který bude obsahovat podstromy i v uzlech označených jako back. Na obr.18 roviny *a,b,c* popisují jednoduché konvexní těleso. Roviny *d,e,f* však tento konvexní mnohostěn dále dělí a vnitřní části mnohostěnu leží na obou stranách dělící nadroviny.



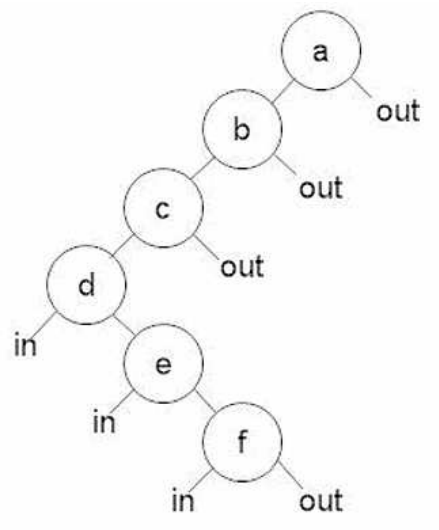

*Obr.18 Rozdělení nekonvexního polygonu na konvexní části polygonu*

Vybereme jednu z nadrovin a umístíme ji do uzlu BSP stromu. Spočítáme vzájemnou polohu této nadroviny a vnitřní oblasti mnohostěnu. Mohou nastat tři varianty:

- Kladná část nadroviny a mnohostěnu nemají žádný společný bod. V tomto případě leží celý zbytek mnohostěnu v uzlu back a uzel front je označen jako vnější. Pokud jsou již zpracovány všechny nadroviny, je uzel back označen jako vnitřní.

- Průnikem kladné části nadroviny a mnohostěnu je celý mnohostěn. Celý mnohostěn proto předáme dále do uzlu front a uzel back je označen jako vnější. Pokud jsou již zpracovány všechny nadroviny je uzel front označen jako vnitřní.
- Nadrovina protíná mnohostěn, takže průnikem mnohostěnu a kladné části nadroviny je neprázdná množina a průnikem záporné části nadroviny a mnohostěnu je rovněž neprázdná množina. Do uzlu front tak předáme průnik kladné části nadroviny a mnohostěnu, do uzlu back zbytek mnohostěnu.

Tento postup opakujeme do té doby než zpracujeme celý mnohostěn.[4]




*Obr.19 BSP strom nekonvexního mnohostěnu*

## 7.3.5. Dijkstrův algoritmus

Tento algoritmus se často použit pro nalezení nejkratší cesty. Pro správnou funkci Dijkstrova algoritmu je zapotřebí ohodnotit všechny hrany grafu nezápornými čísly.

Délka nejkratší cesty ze startovní pozice *S* k vrcholu *v* je označena symbolem *d[v]*. Z toho vyplývá, že vzdálenost *d[S]* je rovna nule. Vzdálenost ze startovní pozice ke zbylým vrcholům, ke kterým ještě nevedou žádné cesty, položíme na začátku rovno ∞. Algoritmus je založen na postupném zpřesňování odhadu délky nejkratší cesty ze startovní pozice k dalším vrcholům.

Princip Dijkstrova algoritmu:

Jestliže $d[u]+w(u,v) < d[v]$, pak $d[u]+w(u,v)$ se stane novým odhadem $d[v]$, kde $(u,v)$ je hrana grafu s ohodnocením $w(u,v)$ a aktuální odhady nejkratších vzdáleností

k vrcholům u,v jsou *d[u], d[v]*. Tento princip se opakuje až do doby, kdy je nalezena cesta do cílové pozice.[4]

### 7.3.6. A* algoritmus

Algoritmus *A** vychází z algoritmu *A*. Jde o algoritmus uspořádaného prohledávání s hodnotící funkcí:

$$f(i) = g(i) + h(i), \qquad [13]$$

kde *g(i)* je cena přechodu z počátečního stavu $s_0$ do stavu *i*, *h(i)* je cena optimální cesty ze stavu *i* do některého z cílových stavů. Cenou rozumíme libovolné nezáporné ohodnocení cesty. Hodnotící funkci *f(i)* lze chápat jako cenu přechodu z počátečního stavu do koncového stavu přes stav *i*.

V mnoha úlohách, ale funkce *g(i)* a *h(i)* neznáme, a proto se používá jejich odhadů $g^*(i)$ a $h^*(i)$.

Funkcí *g(i)* odhadneme minimální dosud zjištěnou cenou $g^*(i)$ přechodu z počátečního stavu do stavu *i*. Funkci *h(i)* nahrazujeme funkcí *h*(i)*, která kvantitativně vyjadřuje náš odhad ceny cesty z uzlu *i* do některého z cílových stavů. Odhad vyjadřuje naši heuristickou znalost o tom, jaké jsou šance nalézt optimální řešení. Funkce $h^*(i)$ je proto nositelem heuristické informace a bývá nazývána heuristickou funkcí. Heuristická funkce je pro efektivnost prohledávání hodně podstatná.

Algoritmus *A* s přípustnou heuristickou funkcí je algoritmus $A^*$. Algoritmus prohledávání je přípustný, jestliže vždy nalezne optimální cestu, pokud tato cesta existuje. Heuristická funkce je přípustná, je–li $h^*(i) \geq 0$ a $h^*(i) \leq h(i)$ pro všechny uzly *i*. To znamená, že je-li funkce *h*(i)* nezáporná a menší nebo rovna skutečné ceně přechodu ze stavu *i* do cílového stavu, je algoritmus *A* přípustný.

Čím je $h^*$ lepším dolním odhadem *h*, tím menší část stavového prostoru se prohledává při hledání optimálního řešení (při $h^* = h$ expanduje algoritmus $A^*$ pouze stavy na cestě k cílovému stavu).

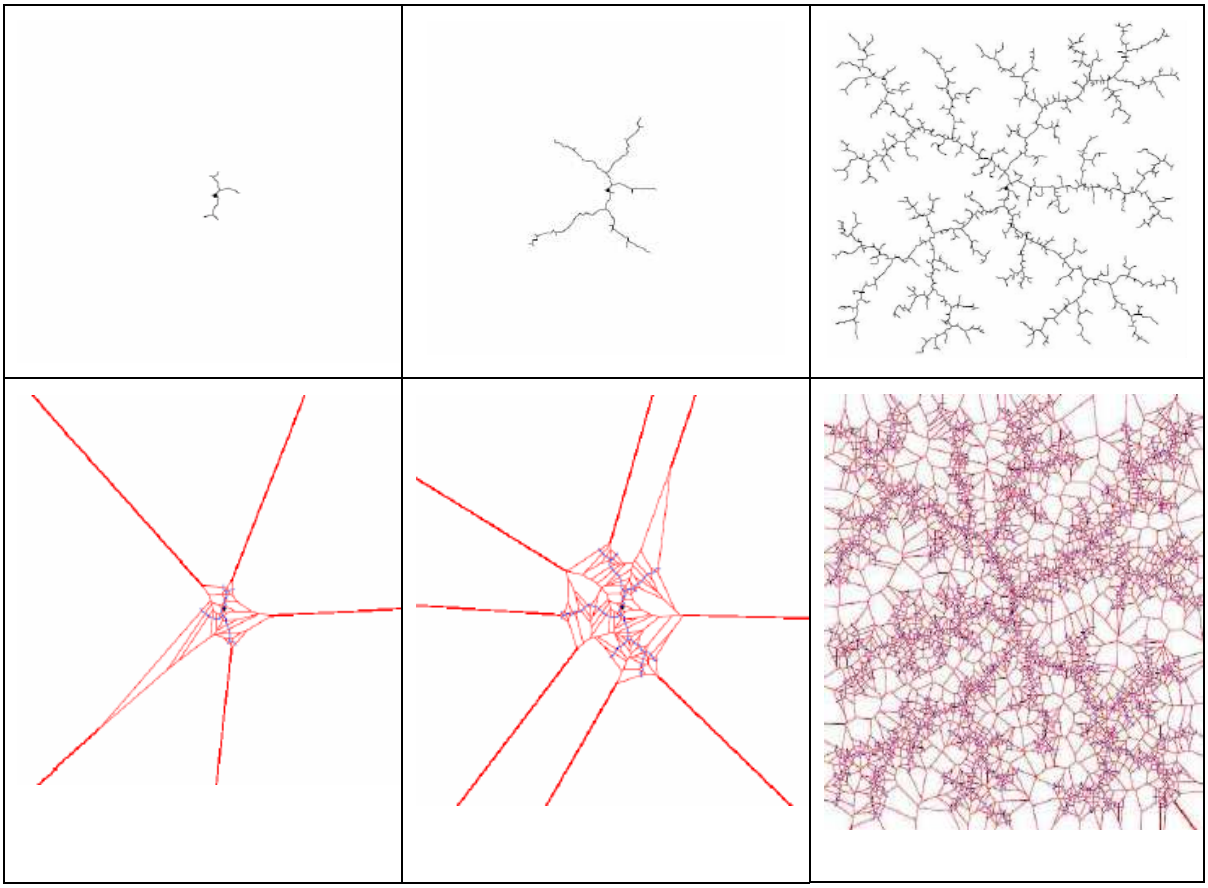
Modifikace algoritmu *A* a *A** rozdělíme na algoritmus se stejnoměrnou cenou a algoritmus větví a mezí.

Funkce $h^*(i)$ je odhad ceny optimální cesty ze stavu *i* do některého z cílových stavů. Algoritmus se stejnoměrnou cenou $h^*(i) = 0$, potom je prohledávání stavového prostoru

řízeno pouze funkcí $g^*(i)$. Je-li nalezen určitý cílový stav, algoritmus si zapamatuje jeho cenu cesty a ze seznamu OPEN vyřazuje uzly s vyšší cenou, než je zapamatovaná cesta řešení. Pokud je nalezen cílový uzel s nižší cenou, pamatuje se nižší cena a algoritmus pokračuje až do okamžiku, kdy je seznam prázdný. Tato modifikace se také nazývá algoritmem větví a mezí.

## 7.4. RRT stromy (Rapidly-exploring Random Trees)

Metoda RRT stromy patří do skupiny pravděpodobnostních plánovacích algoritmů. Poprvé byla představena v roce 1998 Stevem LaVallem. Rychle rostoucí náhodné stromy (RRT) jsou datové struktury a algoritmy pro účinné prohledávání n-dimenzionálního nekonvexního prostoru. Původní návrh byl nejprve navrhnut pro neholonomické plánování cesty robotů s vysokým počtem stupňů volnosti (včetně dynamických omezujících podmínek). RRT je však metoda vhodná pro širokou skupinu problémů plánování cesty. Může se intuitivně považovat za Monte Carlo verzi Vorového diagramu. Principem je vytváření prohledávajícího stromu, který se snaží rychle a rovnoměrně prohledat konfigurační prostor.[5][6][7].



*Obr.20 Horní řádek zobrazuje RRT stromy a dolní řádek zobrazuje Vorového oblasti odpovídající vrcholům RRT stromu*

Před použitím metody RRT je třeba nadefinovat:

- Konfigurační prostor C
- Počáteční konfiguraci $q_{init}$ a cílovou konfiguraci $q_{goal}$, kde $q_{init} \in C$ a $q_{goal} \subset C$
- Pohybové rovnice robotu $\dot{x} = f(x, y)$
- Detektor kolizí je funkce D, která ověřuje polohu náhodné konfigurace $q_{rand} \notin C_{obst}$

  $D : C \rightarrow \{TRUE, FALSE\}$

- Množina vstupů U, která specifikuje vstupy nebo akce, které mohou ovlivnit konfiguraci robota (rychlost, natáčení apod.)
- Inkrementální simulátor, který na základě derivací pohybových rovnic určuje konfiguraci robota v čase t.
- Metriku ρ, kde ρ: X×X→[0,∞] určují vzdálenost dvou konfigurací robotu

### 7.4.1. Základní algoritmus RRT

Následující algoritmus zapsaný v pseudokódu představuje základní verzi RRT stromu, zapsanou v [7].

1. RRT.init ($x_{init}$)
2. repeat
3. for i=1 to CONNECT_CHECK_INTERVAL
4. $x_{rand}$ = random state
5. $x_{closest}$ = GetClosestNode( $x_{rand}$ )
6. $x_{new}$ = GenerateNewNode($x_{closest}$ , $x_{rand}$)
7. $x_{new}$ = ApplyRestrictions($x_{closest}$ , $x_{rand}$)
8. if ($x_{new}$ is OK)
9. RRT.AddNewNode($x_{closest}$ , $x_{new}$)
10. else
11. RRT.Trapped
12. end if
13. end for
14. RRT.TryConnectToGoal
15. until GoalReached

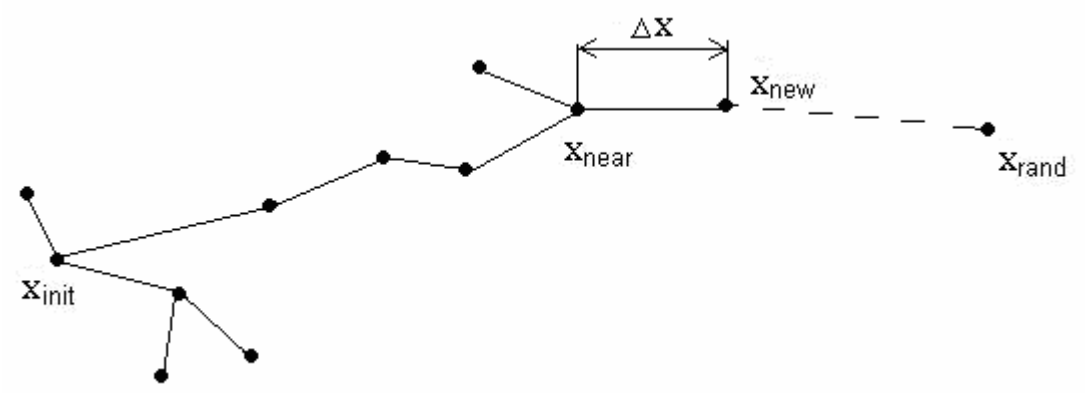
Strom RRT se vytváří iteračně tak, že se v každé iteraci rozroste směrem k náhodně vygenerované konfiguraci robota o nový vrchol. Vrcholy stromu představují konfigurace robotu a jeho hrany akce robotu.

RRT začíná ve startovní pozici $x_{init}$ a během expanze se snaží nalézt cílovou pozici $x_{goal}$. V každém kroku se vygeneruje náhodná konfigurace $x_{rand}$, která leží v konfiguračním prostoru. V průběhu hledání jsou vrcholy stromu tvořeny takovým způsobem, že všechny

vrcholy náleží $C_{free}$. Z tohoto důvodu musí existovat možnost testovat, zda určitý stav leží v kolizním konfiguračním prostoru $C_{obst}$. Stavy v $C_{obst}$ mohou mít v závislosti na aplikaci různý význam. Nejčastěji $C_{obst}$ představuje konfiguraci, kdy je robot v kolizi s překážkou.

V prvním kroku algoritmu je vytvořen základní vrchol ve startovní pozici $x_{init}$. Vygeneruje se náhodná konfigurace $x_{rand}$, která leží v konfiguračním prostoru. Následně se vyhledá nejbližší sousední bod existujícího RRT stromu a realizuje se růst stromu směrem k $x_{rand}$. Nový bod $x_{new}$ je generován ve vzdálenosti $\Delta x$ od nejbližšího sousedního bodu ve směru náhodné konfigurace $x_{rand}$. Dále jsou na nově vytvořený vrchol aplikovány omezující podmínky, to znamená, zda nedošlo ke kolizi s překážkou nebo jestli se nové hrany vrcholu nachází ve volném konfiguračním prostoru $C_{free}$. Pokud jsou tyto podmínky úspěšně splněny přidá se nová konfigurace $x_{new}$ jako nový vrchol do RRT stromu a spojí se hranou s nejbližším vrcholem stromu. Jestliže nový vrchol nové omezení nesplňuje, je jednoduše vyřazen a proces rozrůstání RRT pokračuje. RRT strom se iterativně rozrůstá a algoritmus se snaží spojit vrcholy RRT stromu s cílovou pozicí $x_{goal}$. Hlavní výhodou RRT stromů je rychlé rozrůstání k neprozkoumaným oblastem stavového prostoru. Ohodnocením hrany je vstup $u_{new}$. Mohou nastat tři situace:

- RRT.AddNewNode – nový vrchol $x_{new} \neq x_{rand}$ byl vložen do stromu.
- RRT.Trapped – nepodařilo se nalézt novou konfiguraci ležící ve volném konfiguračním prostoru $C_{free}$.
- GoalReached – nový vrchol $x_{new}$ je totožný s $x_{rand}$. Pro neholomické plánování se netestuje úplná shoda $x_{new}$ s konfigurací $x_{rand}$, ale počítá se vzdálenost mezi $x_{new}$ a $x_{rand}$. Pokud je vzdálenost mezi nimi menší než nějaké $\delta$, je $x_{rand}$ prohlášen za dosažený. Jako $\delta$ se volí malé kladné číslo.




*Obr.21 Rozrůstání RRT stromu*

### 7.4.2. Dvoustromový RRT plánovač

Tento plánovač při prohledávání konfiguračního prostoru používá dva stromy místo jednoho. První strom roste z $x_{init}$ a druhý z $x_{goal}$. Cesta je nalezena, když se oba stromy potkají. Algoritmus dvoustromového RRT plánovače popsaného v [5] vypadá takto:

1. function RRT_BIDIRECTIONAL($x_{init}$, $x_{goal}$)
2. $T_a$.init($x_{init}$); $T_b$.init($x_{goal}$);
3. for k=1 to K do
4. $x_{rand}$ ← RANDOM_CONF();
5. if not (EXTEND($T_a$,$x_{rand}$)=Traped) then
6. if (EXTEND($T_b$,$x_{new}$)=Reached) then
7. Return PATH($T_a$, $T_b$);
8. SWAP($T_a$, $T_b$);
9. Return Failure;

Funkce RRT_BIDIRECTIONAL rozděluje výpočet na dvě části. První část výpočtu tvoří prohledávání konfiguračního prostoru a druhou snaha o spojení dvou stromů. V každé iteraci se jeden strom rozroste o nový vrchol $x_{new}$ směrem k náhodné konfiguraci $x_{rand}$ a druhý strom se pokouší napojit na nový vrchol $x_{new}$ v prvním stromu. Potom si oba stromy vymění role a výpočet se opakuje. Cesta je nalezena, jakmile dojde ke spojení obou stromů. Velkým problémem tohoto plánování je propojení obou stromů v jeden, především pak u neholomického plánování.

### 7.4.3. Srovnání jednostromové a dvoustromové metody RRT

Při srovnávání obou metod RRT algoritmů musí být stejné počáteční podmínky, to znamená stejná mapa prostředí. Parametr, na kterém nejvíce závisí rychlost vyhledávání je vzdálenost kroku $\Delta x$.

Výhodou dvoustromové metody je, že dochází k významnému snížení počtu vrcholů při maximální možné vzdálenosti kroku $\Delta x$. To má za následek významné zlepšení rychlosti vyhledávání.

U dvoustromové metody jsou dvě významné nevýhody oproti jednostromové metodě. První nevýhoda souvisí s paměťovými nároky. Celková doba prohledávání je dána počtem vrcholů obou stromů. Je zapotřebí, aby si algoritmus pamatoval, které vrcholy se již snažil spojit. Tím dochází k vyšším paměťovými nárokům. A proto pro zvýšení rychlosti vyhledávání je potřeba mít přídavnou paměť.

Druhá nevýhoda dvoustromového plánování je přesné propojení obou stromů. Pro jednostromové neholomické plánování se nezkouší úplná shoda vrcholů, ale počítá se vzdálenost. Pokud je vzdálenost menší než nějaké zadané malé kladné číslo je tento vrchol prohlášen za dosažený. U dvoustromového plánování je přesné propojení obou stromů dosti nepravděpodobné. Proto zde musí být speciální aplikace nazvaná jako blízký vrchol. Pro neholomické plánování tato aplikace urychlí celý proces vyhledávání.

Dvoustromová metoda RRT algoritmu přinese značné zmenšení množství vrcholů ve výsledné stromové struktuře a tím dojde ke snížení doby prohledávání. V závislosti na zvláštní aplikaci se doba prohledávání také snižuje a může přinést podstatné zlepšení celé aplikace.

### 7.4.4. Vlastnosti RRT

RRT algoritmus má několik vlastností, které jej činí ideálně vhodným nástrojem pro řešení širokého množství praktických problémů plánování cesty.

- Strom má silnou tendenci růst směrem k dosud neprozkoumaným oblastem.
- RRT je za obecných podmínek pravděpodobnostně kompletní. Pokud existuje cesta tak pravděpodobnost, že bude nalezena, konverguje k jedné jak se počet iterací blíží k nekonečnu.
- RRT algoritmus je poměrně jednoduchý, což ulehčuje provedení vyhodnocení.

- Vrcholy stromu zůstávají vždy propojeny, i když je počet hran minimální.
- RRT může být uvažován jako samostatný plánovací modul, který lze přizpůsobit a začlenit do jiných plánovacích systémů.
- Celý plánovací algoritmus může být konstruován bez potřeby schopnosti řídit systém mezi dvěma určenými stavy, což značně rozšiřuje jeho aplikovatelnost.

# 8. IMPLEMENTACE

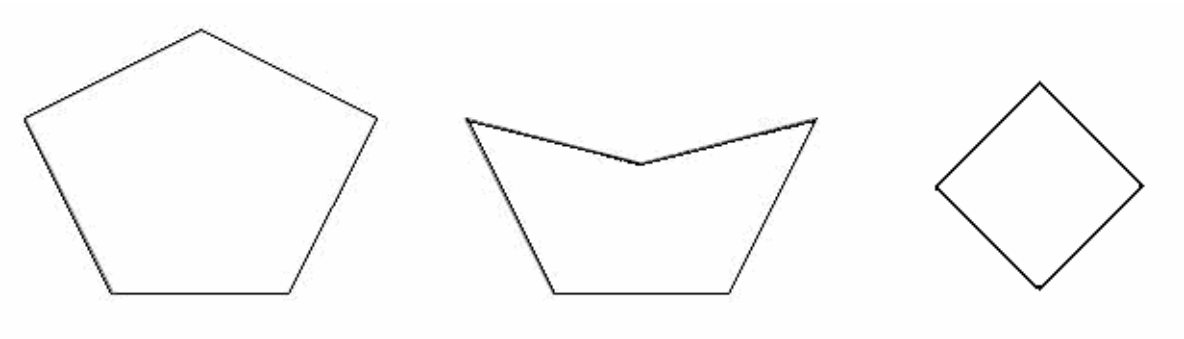
Cílem diplomové práce je navrhnout modul plánování trajektorie robotu ve známé mapě. Jedná se o nalezení bezkolizní cesty z počáteční do cílové pozice ve známé mapě. Mapa je tvořena dvoudimenzionálním pracovním prostorem robotu, který obsahuje statické geometrické překážky.

Modul plánování cesty využívá mapy pracovního prostoru robotu a umožňuje plánování cesty algoritmem RRT. Aby navržený modul bylo možné použít v dalších programovacích jazycích je potřeba zvolit univerzální komunikační rozhraní. Jako ideální univerzální rozhraní je zvolena dynamická knihovna DLL. Toto rozhraní umožňuje přístup k plánovacím funkcím zcela nezávisle na použitém plánovacím algoritmu.

Modul plánování cesty je napsán v programovacím prostředí Delphi. V tomto programu je použit a pozměněn laboratorní kód pro metodu RRT[10]. Další experimenty byly vyzkoušeny v programovacím prostředí jazyka C a v prostředí LabView od firmy National Instruments.

## 8.1. Překážky a detekce kolizí

V unitě WORLD jsou popsány překážky a možné kolize s těmito překážkami. Překážky jsou tvořeny geometrickými prvky, jako jsou přímky, čtverce, obdélníky a jednoduché polygony. Ukázky jednotlivých geometrických prvků jsou zobrazeny na obr.22.



*Obr.22 Jednoduché polygony (konvexní a konkávní) a čtverec*

Detekce kolize s překážkou je provedena tak, že se z bodu $x_{new}$ ve směru náhodně vygenerovaného bodu $x_{rand}$ vygeneruje bod, který má souřadnice x a y. Tento bod se poté porovnává se souřadnicemi každé překážky. Jestliže je vzdálenost souřadnic menší nebo rovna vzdálenosti souřadnic překážky tak je tento stav nazván jako kolizní. Jakmile se

vyhoví dané podmínce, může se strom o náhodný bod rozšířit, protože vzniklý bod bude ležet ve volném prostoru $C_{free}$ a nebude kolidovat s žádnou překážkou.

## 8.2. RRT algoritmus

Pro plánování cesty se používá jednostromová RRT metoda. Důvodem, proč je použita jednostromová metoda RRT je takový, že v případě nehomologického robotu by bylo problematické napojení obou stromů v místě jejich setkání.

Následující implementace algoritmu RRT metody představuje základní verzi RRT stromu.

Implementovaný algoritmus RRT

- Function RRT ($x_{init}$, $x_{goal}$)
- Jako kořen stromu se vezme $x_{init}$
- Opakuj cyklus iterací

1. Vygeneruj náhodnou konfiguraci vrcholu $x_{rand}$
2. Najdi vrchol $x_{near}$ který je nejblíže vrcholu $x_{rand}$
3. Vypočítej nový vrchol $x_{new}$ z konfigurací vrcholů $x_{near}$ a $x_{rand}$
4. Přidej nový vrchol $x_{new}$ do stromu
5. Spoj $x_{new}$ a $x_{near}$ hranou stromu
6. Zkus, jestli lze jít z nového vrcholu $x_{new}$ do cílového vrcholu $x_{goal}$, pokud ano ukonči cyklus, cíl nalezen

- Pokud není cíl dosažen opakuj body 1 až 6 než je nalezena cílová pozice $x_{goal}$
- Najdi cestu stromem od $x_{goal}$ do $x_{init}$

Algoritmus RRT je sestaven tak, že nejdříve je zadána startovní pozice $x_{init}$ a cílová pozice $x_{goal}$ ve známé mapě. Poté je vytvořen cyklus pro vytvoření RRT stromu. Prvním bodem cyklu je generování úplně náhodné pozice $x_{rand}$. Tato pozice může ležet kdekoliv v konfiguračním prostoru C. Následuje nalezení vrcholu $x_{near}$ který leží nejblíže vrcholu $x_{rand}$. Dále je snaha nalézt nový vrchol $x_{new}$ z konfigurací vrcholů $x_{near}$ a $x_{rand}$. Provede se to tak, že se hledá minimální vzdálenost mezi $x_{near}$ a $x_{rand}$. Nová konfigurace $x_{new}$ musí ležet ve volném konfiguračním prostoru. Stav konfigurace $x_{new}$ je kolizní pokud je vzdálenost souřadnic $x_{rand}$ menší nebo rovna vzdálenosti souřadnic překážky. Naopak nekolizní stav

vyhovuje dané podmínce a strom se může rozrůst o nový vrchol. Po nalezení nového vrcholu se hranou spojí vrcholy $x_{near}$ a $x_{new}$. Cesta z $x_{init}$ do $x_{goal}$ je nalezena, jestliže je vzdálenost mezi $x_{new}$ a $x_{goal}$ menší než předem zadaná vzdálenost od cíle $\delta$. Tato vzdálenost do cíle $\delta$ se uvádí, protože je v podstatě nemožné dosáhnout přesné shody $x_{new}$ s $x_{goal}$ v rozumném čase.

Cesta stromem ze startovní pozice $x_{init}$ do cílové pozice $x_{goal}$ se vyhledává v opačném směru od $x_{goal}$ do $x_{init}$, protože při stromové struktuře je nalezení cesty tímto způsobem jednodušší a rychlejší než ve směru od $x_{init}$ do $x_{goal}$.

## 8.3. Nastavení parametrů RRT algoritmu

Pro nastavení parametrů RRT algoritmu byly vytvořeny patřičné procedury a pro zapsání globálních proměnných metody RRT slouží unita GLBLS. Jako globální proměnné jsou v unitě uvedený:

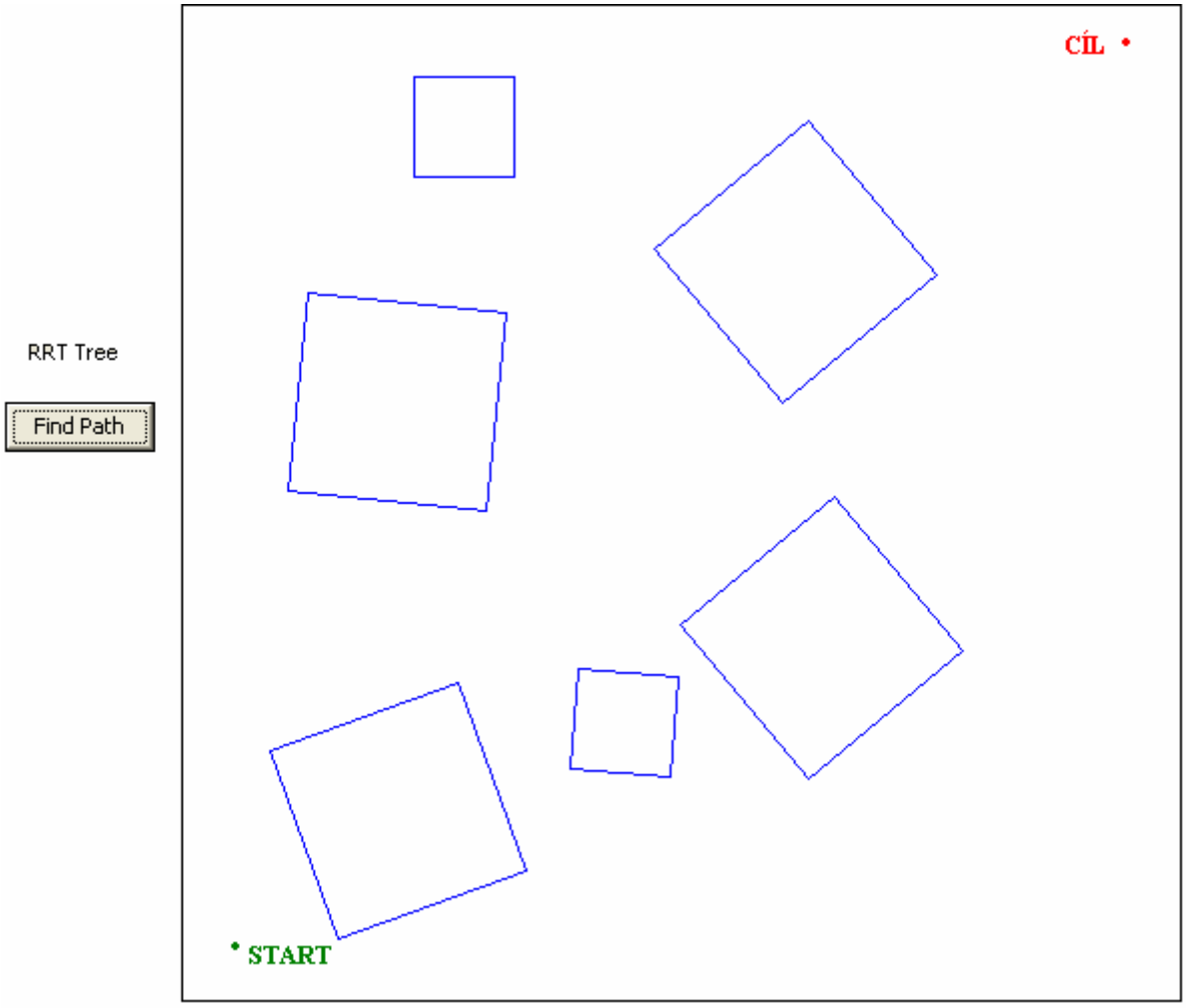
- Maximální počet kroků (iterací)
- Po kolika iteracích zkusit cílovou pozici
- Délka kroku
- Cílová vzdálenost – vzdálenost od cíle, která se již považuje za cíl

Pro nastavení ostatních parametrů slouží procedury, které jsou vytvořeny tak, aby mohly být dále použity i pro rozhraní DLL knihovny. Tyto procedury jsou:

- Procedura initRRTInit
- Procedura vytvorSetStart
- Procedura vytvorSetGoal
- Procedura RRTFindPath
- Procedura VytvorSvet
- Procedura NakresliSvet
- Procedura VyberVykresleni

Procedura initRRTInit je určena pro načtení nezávislých proměnných z unit RRT a WORLD. V proceduře vytvorSetStart je provedeno nastavení startovní pozice $x_{init}$ pro vyhledávání ve známé mapě. Startovní pozice může být zvolena na libovolném volném

místě v mapě, to znamená, že se nesmí umístit do překážky. Nastavení startovní pozice se provede pomocí zadání souřadnic x a y. Procedura vytvorSetGoal je opakem procedury vytvorSetStart. Pomocí této metody je nastavena cílová pozice $x_{goal}$ a platí pro její nastavení stejné požadavky jako při nastavení startovní pozice. Na obr.23 je ukázána mapa a v mapě je již přidána startovní i cílová pozice pro hledání.

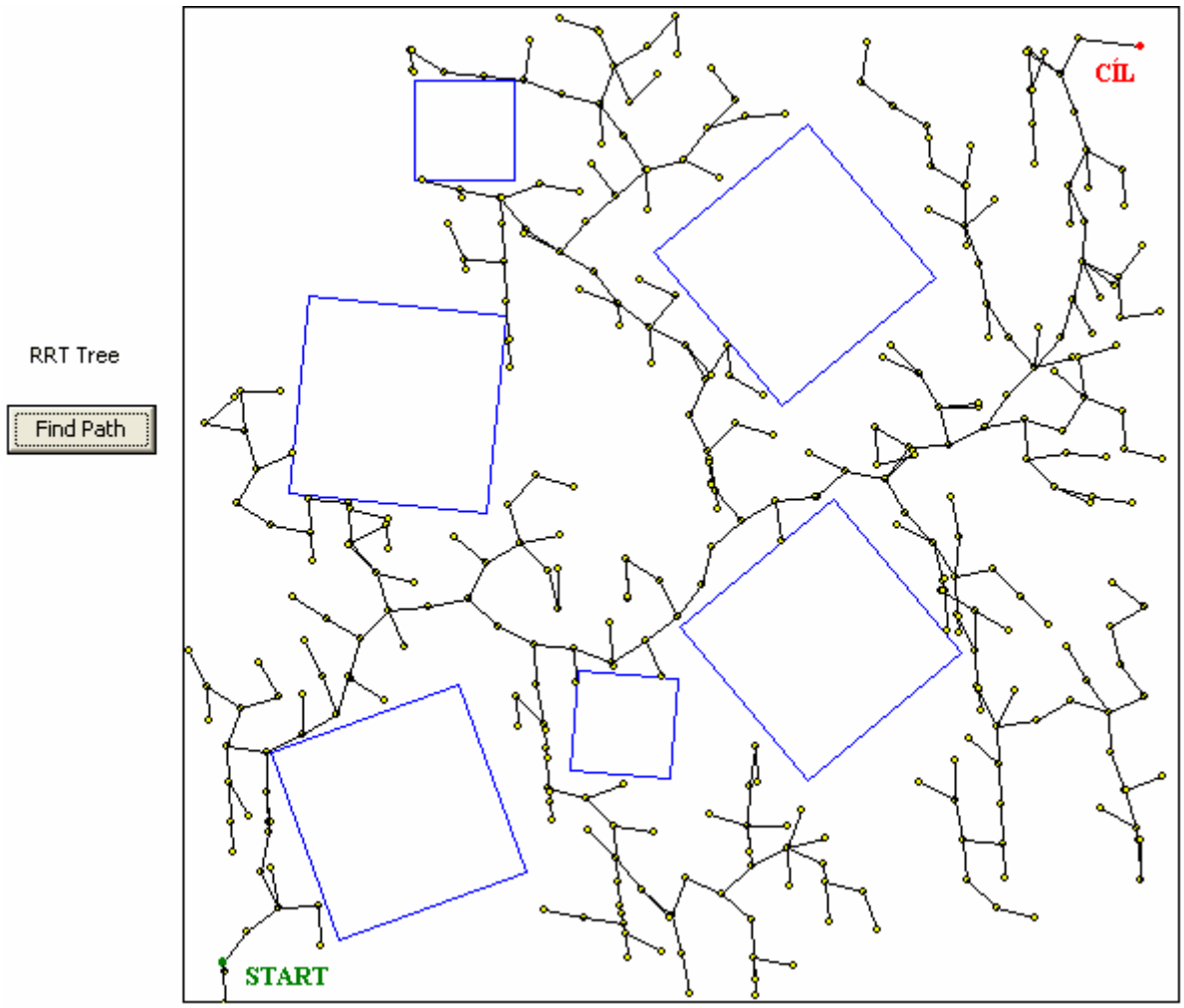



*Obr.23 Startovní a cílová pozice ve známé mapě*

Velice důležitou procedurou je RRTFindPath. Tato procedura reprezentuje možné nastavení maximálního počtu kroků(iterací). Počet iterací se musí zvolit dostatečný k velikosti dané mapy, aby se prohledala celá mapa a došlo k nalezení cíle. V opačném případě by mohlo dojít k prohledání jen části mapy a tím by nebyla nalezena cesta od startu do cíle. Dalším nastavením je po kolika iteracích se má zkusit cílová pozice. Důležitým přednastavením je délka kroku. Je-li zadaná délka kroku malá může se stát, že se nenalezne cesta k cíli při uvedeném maximálním počtu iterací. Naopak při nastavení velké délky

kroku může dojít k tomu, že bude délka kroku větší než je velikost nejmenší překážky, což by způsobilo nalezení cesty přes překážku a tím by samozřejmě došlo k nalezení nesprávné (kolizní) cesty. Posledním z nastavení RRT metody se určí vzdálenost od cíle δ, která se již považuje za cíl. Pro neholomické plánování se netestuje úplná shoda $x_{new}$ s konfigurací $x_{rand}$, ale počítá se vzdálenost mezi $x_{new}$ a $x_{rand}$. Pokud je vzdálenost mezi nimi menší než zadané δ, je $x_{rand}$ prohlášen za dosažený. Jako δ se volí malé kladné číslo.

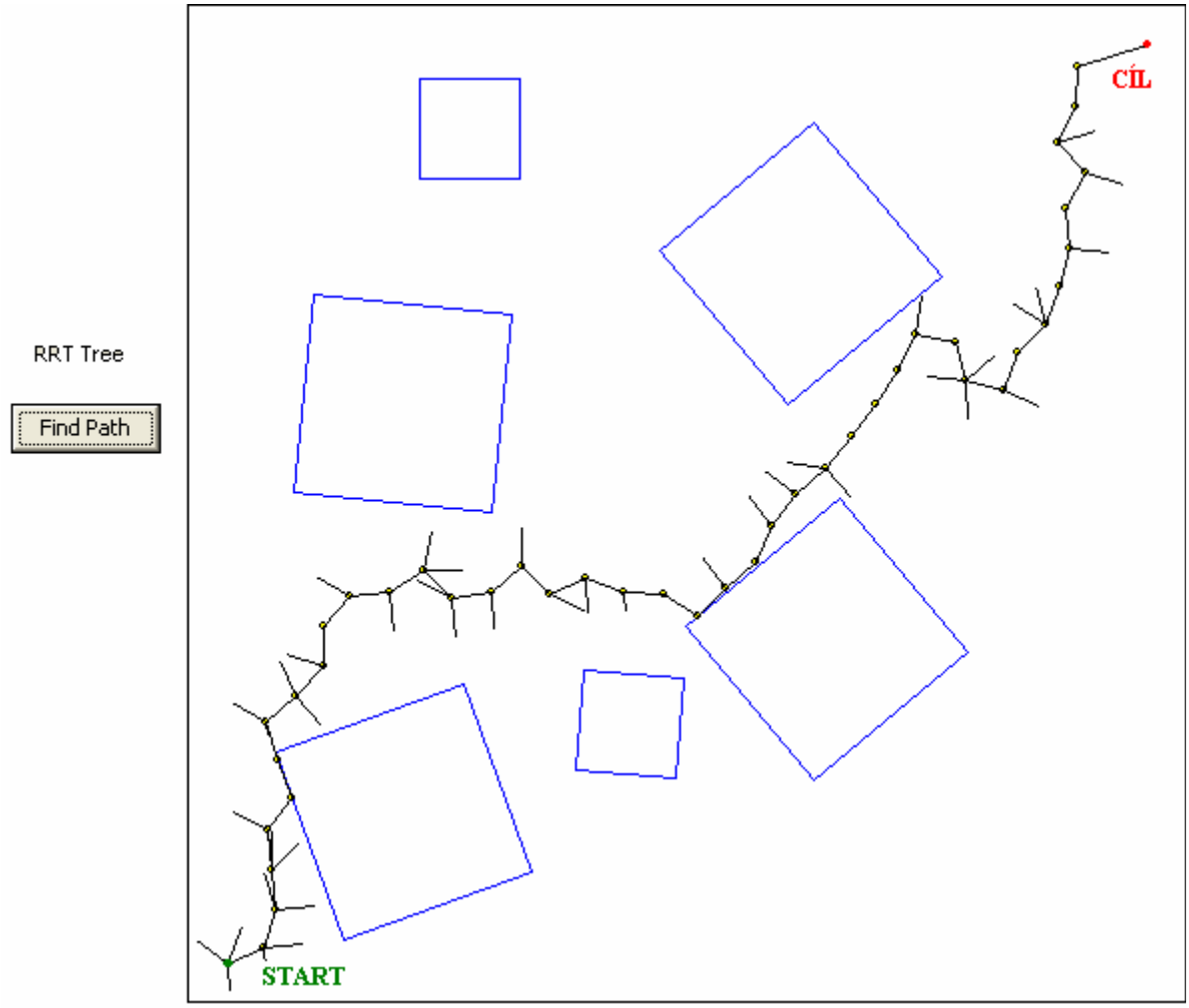
V proceduře VytvorSvet dojde k načtení mapy a v této mapě je provedeno vykreslení RRT stromu. Mapa prostředí je předem známá a musí být předem přidána do programu. Při výběru vykreslení je možné volit z vykreslení cesty ze startovní pozice do cílové pozice nebo vykreslení všech vrcholů, které byly vytvořeny při hledání. Na obr.24 je ukázáno vykreslení všech celého stromu při prohledání známé mapy prostředí.




*Obr.24 Vykreslení RRT stromu*

Pokud je zvolena možnost vykreslení cesty ze startovní pozice $x_{init}$ do cílové pozice $x_{goal}$ je důležité si uvědomit, že se jedná o cestu stromem. Je proto jednodušší postupovat v obráceném směru od $x_{goal}$ do $x_{init}$. V tomto případě, kdy je strom prohledáván opačným směrem je nalezení cesty rychlejší a jednodušší, než kdyby bylo postupováno od startovní pozice $x_{int}$ do cílové pozice $x_{goal}$. Výslednou cestu tvoří seznam konfigurací $X_{path} = \{x_1, x_2, \ldots, x_n\}$ a seznam hran $U_{path} = \{u_1, u_2, \ldots, u_n\}$.

Funkčnost celého vyhledávání je vztaženo na jedno tlačítko FindPath a všechny zmíněné procedury jsou vnořeny do tohoto tlačítka. Proto se nastavení výše uvedených parametrů provádí v kódu programu.




*Obr.25 Nalezená cesta ze startovní pozice do cílové pozice*

## 8.4. Univerzální komunikační rozhraní

Jako univerzální komunikační rozhraní je zvolena dynamická linkovaná knihovna (DLL). DLL knihovny jsou vlastně soubory, ve kterých jsou uloženy procedury a funkce, které může využívat i více běžících aplikací najednou.

Základní znaky dynamických knihoven jsou:

- Knihovny DLL a jejich obsah, tj. programový kód a data, se k aplikaci připojují (linkují) dynamicky, tj. za jejího běhu, a to až v okamžiku, kdy je to potřeba.
- Knihovny DLL se mohou sdílet, tzn. že obsah téže DLL knihovny může používat více aplikací.

Výhody DLL knihoven jsou takové, jakmile se požaduje provést změna ve funkčnosti aplikace, není nutné aktualizovat a kompilovat celý program, ale stačí nahradit jednu verzi knihovny za novou verzí. Další výhoda je, že jednotlivé procedury a funkce obsažené v DLL knihovně je možné načítat do paměti až v okamžiku, kdy je zapotřebí. Pokud procedura nebo funkce z DLL knihovny již není potřeba, je možné ji z paměti uvolnit. Hlavní výhodou DLL knihoven je nezávislost na programovacím jazyku. Například knihovnu vytvořenou v programovacím jazyce Delphi je možné bez větších problému používat v aplikacích vytvořených v jiných programovacích jazycích. Nevýhodou DLL knihoven je o trochu pomalejší přístup k procedurám a funkcím.

V praktické části jsou v dynamické knihovně zapouzdřeny unity RRT, WORLD, GLBLS. Z těchto tří unit je zapotřebí vybrat potřebné procedury pro univerzální komunikační rozhraní. Tím je dosáhnuto toho, že je možné se dostat k plánovacím funkcím (procedurám) nezávisle na použitém programovacím jazyce. Tyto procedury jsou:

- Procedura initRRTInit
- Procedura vytvorSetStart
- Procedura vytvorSetGoal
- Procedura RRTFindPath
- Procedura VytvorSvet
- Procedura NakresliSvet
- Procedura VyberVykresleni

Pomocí těchto procedur je možné nastavovat všechny parametry jako při nastavení parametrů RRT algoritmu. To znamená, že je možné nastavovat startovní a cílovou pozici, maximální počet iterací, délku kroku a vzdálenost cíle, která se již považuje za cíl.

Pro získání vykreslení cesty ze startovní do cílové pozice je nutné znát souřadnice jednotlivých bodů stromu. Tyto souřadnice získáme tak, že při spuštění vyhledávání cesty se automaticky vytváří soubor *data.txt*, který obsahuje základní informace o vyhledané cestě. Jeho formát je definován tak, aby informace byly snadno zpracovatelné v programu MS Excel. Jedná se o textový soubor s následujícím uspořádáním dat:

466;457;426

447;452;426

449;432;426

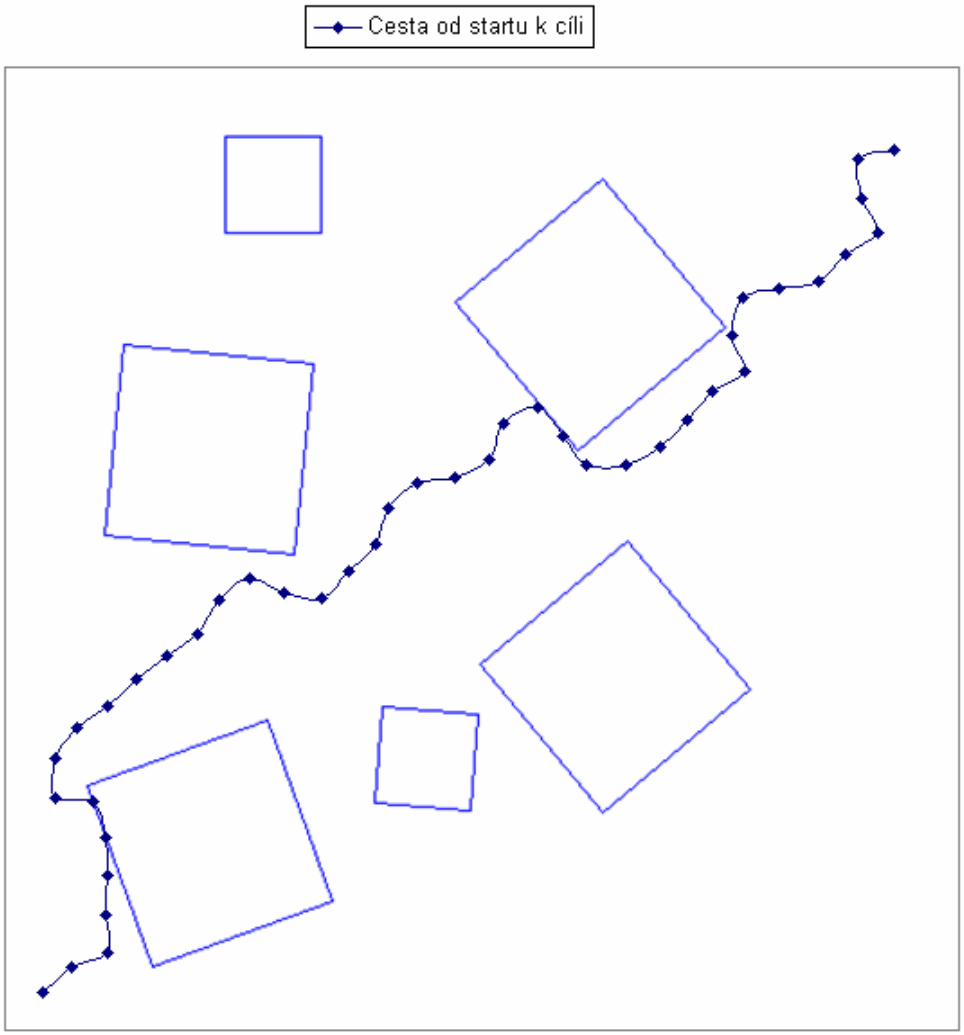
458;414;426

441;403;426

…

Jsou v něm tři sloupce oddělené středníky. Každý řádek odpovídá jednomu vrcholu hledaného stromu. První sloupec označuje x-ovou souřadnici vrcholu a druhý sloupec y-ovou souřadnici vrcholu. Ve třetím sloupci je výsledný počet vrcholů RRT stromu, než byla nalezena výsledná cesta. V tomto okamžiku je již možné provést vykreslení cesty ze startovní pozice do cílové pozice.

Vykreslení se provede pomocí importování dat z textového souboru do MS Excelu. Soubor se jednoduše načte do Excelu a jako oddělovač sloupců se nastaví středník. Tím, že dopředu známe tvar mapy je možné pomocí grafu ukázat vykreslenou cestu ve známé mapě.




*Obr.26 Nalezená cesta v MS Excel*

## 8.5. Ověření funkčnosti navržené dynamické knihovny

Ověření funkčnosti dynamické knihovny vytvořené v programovacím jazyce Delphi bylo provedeno v programovacím jazyce C a v prostředí LabView. V obou případech po přidání dynamické knihovny bylo možné nastavovat všechny parametry RRT algoritmu. Vykreslení výsledné cesty je za pomocí získaných souřadnic vrcholů opět provedeno v MS Excel.

### 8.5.1. Programovací jazyk C

Ověření funkčnosti v programovacím jazyce C je provedeno tak, že se nejdříve musí nadeklarovat funkce, které jsou používané v dynamické knihovně. Poté dojde k samotnému načtení dynamické knihovny a následuje ošetření zdá byla daná knihovna správně přidána, aby se s ní mohlo dále pracovat.

Dalším krokem je získání odkazu na funkce a následné přetypovaní těchto funkcí. Po přetypování dochází k volání daných funkcí. Při volání funkcí se nastavuje startovní a cílová pozice a další parametry RRT algoritmu.

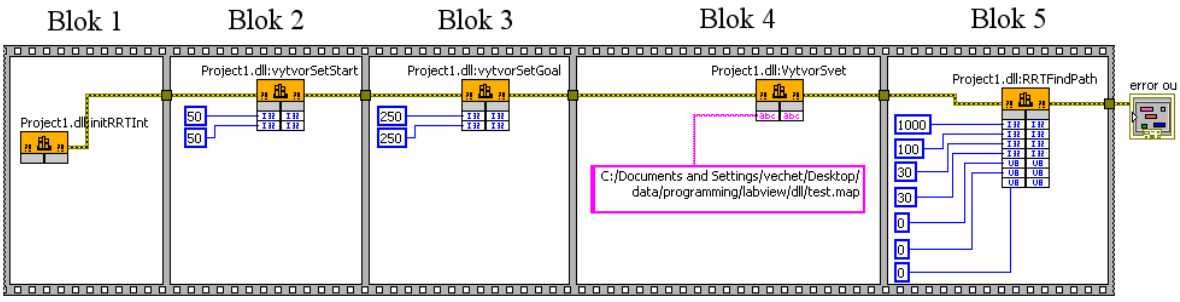
Ukázka vytvořeného algoritmu:

```
//deklarace fci pouzivanych v dll knihovně
typedef int (WINAPI * PRVNI_INIT)(void);
typedef int (WINAPI * DRUHA_START)(int, int);
typedef int (WINAPI * TRETI_GOAL)(int, int);
typedef void (WINAPI * CTVRTA_NACTENI)(void);
typedef int (WINAPI * PATA_RRT)(int, int, int, int, int, int, int);
    int main () {
    HINSTANCE hdll;
//nacteni knihovny
    hdll = LoadLibrary("Project1.dll");
//osetreni nacteni
    if(hdll == NULL){
    printf("dll je spatna!\n");
    return GetLastError();
    }
//ziskani odkazu na fce a jejich pretypovani
PRVNI_INIT        prvni_init      =(PRVNI_INIT)GetProcAddress(hdll, "initRRTInt");
DRUHA_START       druha_start     =(DRUHA_START)GetProcAddress(hdll, "vytvorSetStart");
TRETI_GOAL        treti_goal      =(TRETI_GOAL)GetProcAddress(hdll, "vytvorSetGoal");
CTVRTA_NACTENI    ctvrta_nacteni =(CTVRTA_NACTENI)GetProcAddress(hdll, "VytvorSvet");
PATA_RRT          pata_rrt        =(PATA_RRT)GetProcAddress(hdll, "RRTFindPath");
```

```
//volani danych fci
prvni_init();
druha_start(50, 50);              //zadani startovni pozice
treti_goal(250, 250);             //zadani cilove pozice
ctvrta_nacteni(./test1.map);      //nacteni mapy
pata_rrt(1000,100,30,30,0,0,0);//zadani parametru RRT
        printf("vsechno ok, knihovna nactena \n");
        system("PAUSE");
        return 0;
```

### 8.5.2. Vývojové prostředí LabView

Další ověření funkčnosti je provedeno ve vývojovém grafickém prostředí LabView od společnosti National Instruments. V tomto vývojovém prostředí se vytváří aplikace pomocí tzv. virtuálních přístrojů (Virtual Instruments), kterými se měří a zpracovávají naměřená data. Při tvorbě aplikace virtuálního přístroje se pracuje ve dvou základních oknech, označených jako Front panel a Block diagram. Ve front panelu je vytvářen vnější vzhled přístroje, tzn. rozmístnění prvků, jejich vzhled atd.. Ve druhém okně je modelována aplikace jako blokové schéma algoritmu.



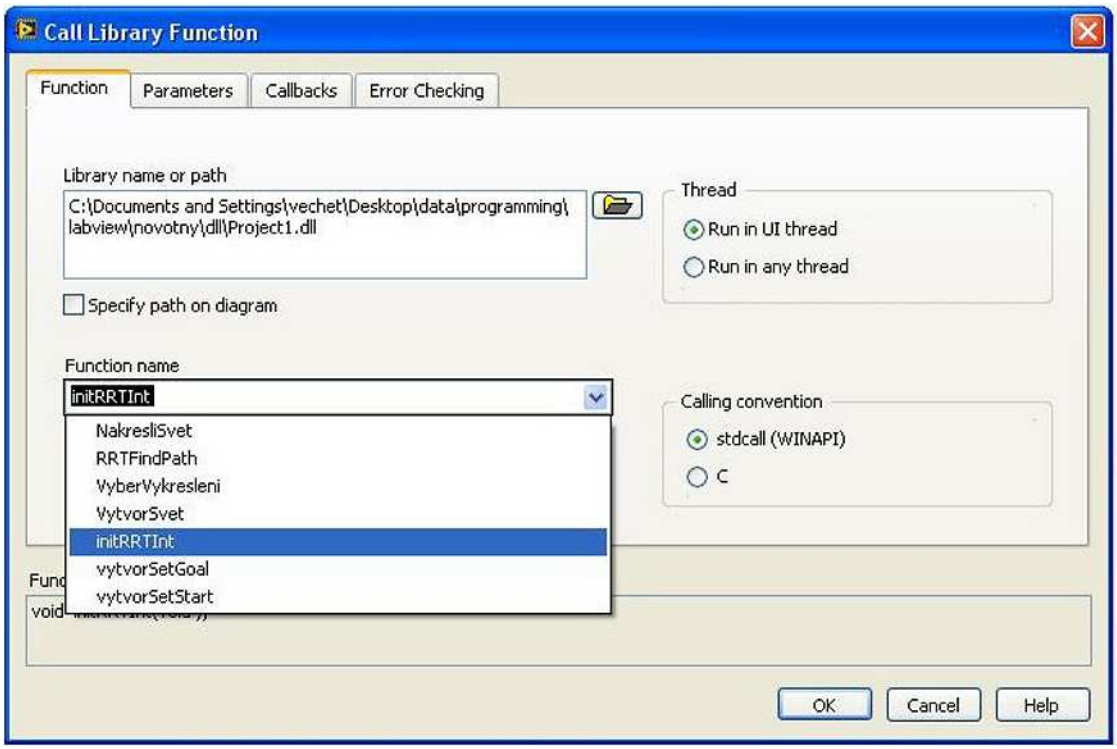

*Obr.27 Blokový diagram*

Pro načtení vytvořené dynamické knihovny je použita knihovna Advenced z široké nabídky knihoven LabView. Knihovna Advenced umožňuje volání funkcí z vytvořené dynamické knihovny. Blokové schéma je znázorněno na obr.27.

Blokový diagram je sestaven z pěti bloků a v každém bloku se nastavuje cesta k dynamické knihovně. K jednotlivým blokům se přiřadí dané funkce, pomocí kterých je

možno nastavovat parametry dynamické knihovny. Každý blok představuje jednu funkci z rozhraní dynamické knihovny.

- Blok 1 – slouží pro nastavení funkce initRRTInit, která je určena pro načtení nezávislých proměnných v dynamické knihovně
- Blok 2 – slouží pro nastavení funkce vytvorSetStart, ve které se zadává startovní pozice pro vyhledávání ve známé mapě. Nastavení startovní pozice se provede pomocí zadání souřadnic x a y
- Blok 3 – slouží pro nastavení funkce vytvorSetGoal, která udává cílovou pozici a je zadána taktéž souřadnicemi x a y
- Blok 4 – slouží pro načtení známé mapy pro hledání cesty mezi startovní a cílovou pozicí
- Blok 5 – slouží pro nastavení RRT algoritmu

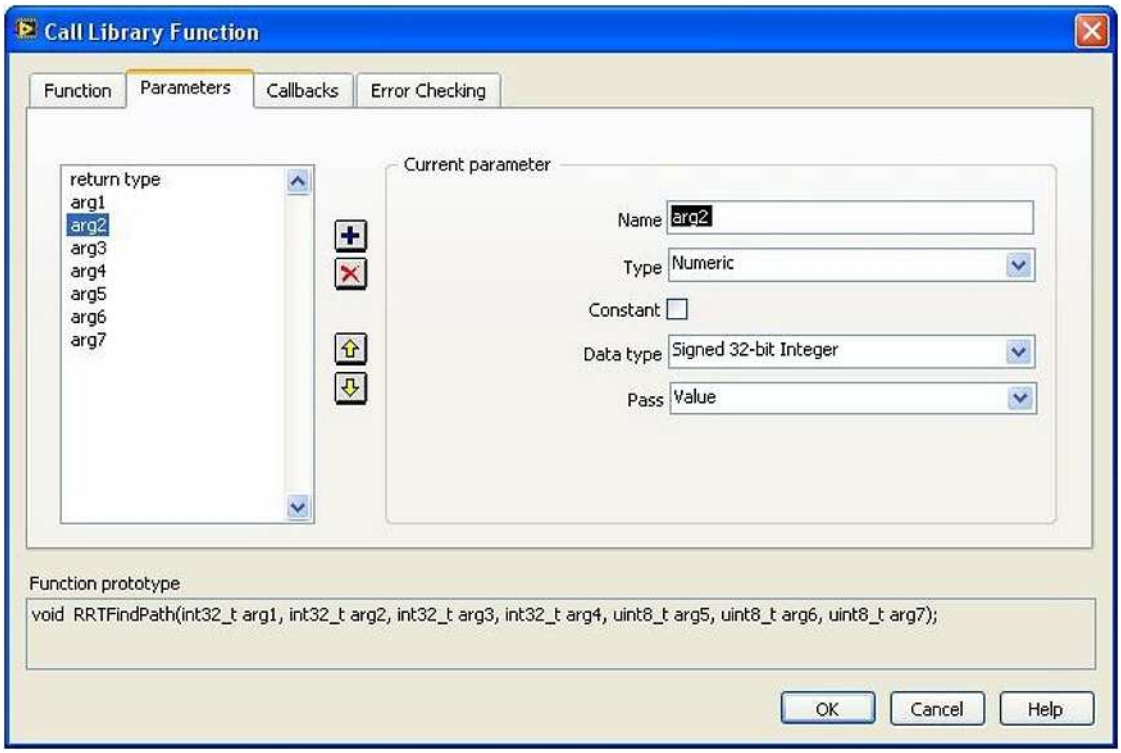



*Obr.28 Načtení dynamické knihovny a nastavení funkce*

V jednotlivých blocích se provádí konkrétní nastavení daného bloku. Na obr.29 je ukázka nastavení bloku 1. Nejprve je zadána dynamická knihovna, se kterou se bude pracovat. Je nutné označit funkci, se kterou bude daný blok pracovat. Dále se v nabídce

volací posloupnost zaškrtne stdcall. Tato volací posloupnost zajišťuje správné načtení funkcí.

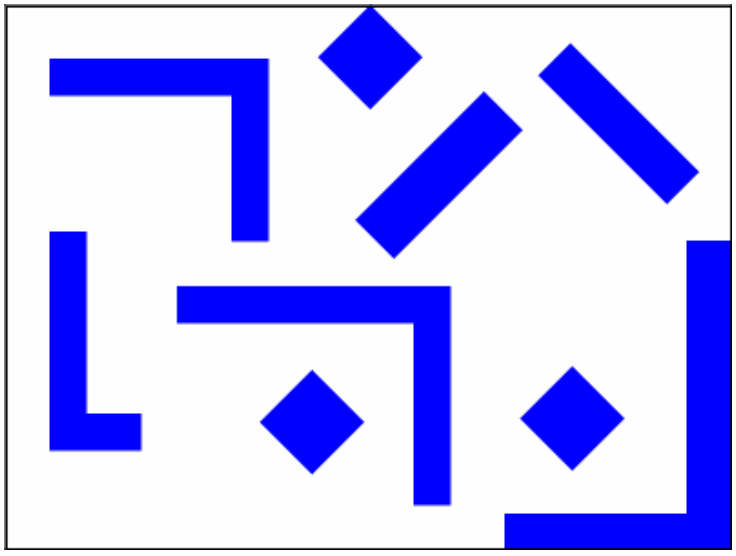
Při nastavení parametrů se nastaví jméno a typ parametru. Poté je potřeba zadat datový typ parametru tak, aby byl dostatečně velký při zadávání parametrů.



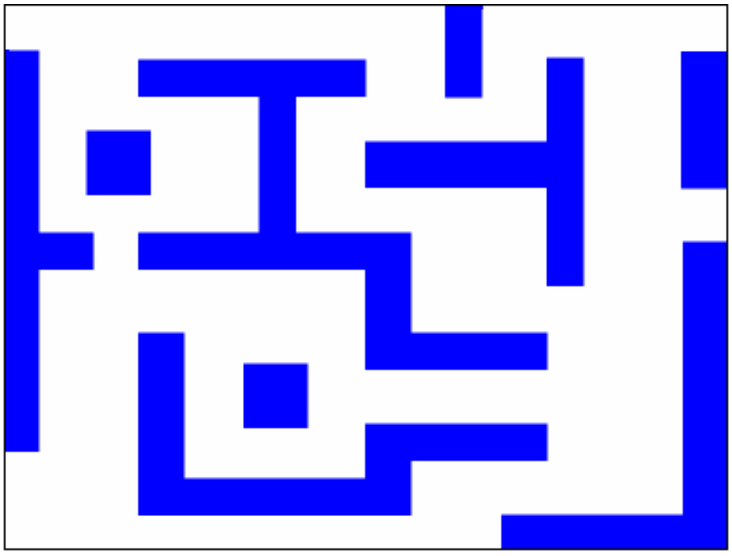

*Obr.29 Nastavení jednotlivých parametrů*

# 9. EXPERIMENTY

Při experimentech bylo možné sledovat chování algoritmu a ověřit vlastnosti tohoto algoritmu. Aby bylo možné pozorovat chování a vlastnosti algoritmu byly vytvořeny dvě mapy pracovního prostředí různé složitosti. První mapa představuje jednodušší mapu a je zobrazena na Obr.30. Složitější mapa připomíná bludiště a je ukázána na obr.31. Složitost jednotlivých map se posuzuje z hlediska počtu a složitosti překážek.



*Obr.30 Jednodušší mapa*



*Obr.31 Složitější mapa*

Prvním experimentem je porovnání celkového počtu vyhledaných vrcholů při nalezení cesty ze startovní pozice do cílové pozice. Startovní a cílové pozice jsou v obou mapách zvoleny stejně. To znamená, že startovní pozice v jednodušší mapě má stejné souřadnice x a y jako startovní pozice ve složitější mapě. To stejné platí i pro cílovou pozici. Počet nalezených vrcholů je závislý na členitosti prohledávané mapy.

Druhým experimentem je porovnání počtu vrcholů potřebných pro vyhledání nejkratší cesty ze startovní do cílové pozice. U nalezení nejkratší cesty se využívá stromové struktury, kdy se prochází vrcholy od cílové pozice do startovní pozice.

Třetím experimentem je porovnání počtu nalezených vrcholů při zadání různé délky kroku. Délka kroku má vliv na počet prohledaných vrcholů. Je-li délka kroku kratší tak počet prohledaných vrcholů je vyšší a může dojít k nenalezení cesty z důvodu vyčerpání zadaného počtu iterací. Při větší délce kroku není prohledání pracovního prostoru tak důkladné, ale nedojde k vyčerpaní zadaného počtu iterací. Při zadání velmi dlouhého kroku může dojít k tomu, že délka kroku bude větší než je délka nejmenší překážky a tím by mohla nastat situace, kdy bude nalezena cesta, avšak tato cesta bude vést přes překážku a tudíž bude kolizní.

Na obr.32 je porovnání nalezených vrcholů při zadané různé délce kroku u jednodušší mapy.

| Celkový počet vrcholů a počet vrcholů při nalezení nejkratší cesty při různé délce kroku | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Číslo pokusu | Jednodušší mapa s délkou kroku 15 | | | | Jednodušší mapa s délkou kroku 30 | | | |
| | Počet vrcholů | | Průměrný počet | | Počet vrcholů | | Průměrný počet | |
| | Celkem | Nejkratší cesta | Celkem | Nejkratší cesta | Celkem | Nejkratší cesta | Celkem | Nejkratší cesta |
| 1 | 1239 | 111 | 1226 | 105,4 | 572 | 53 | 605 | 53,2 |
| 2 | 1030 | 104 | | | 881 | 59 | | |
| 3 | 1471 | 102 | | | 418 | 50 | | |
| 4 | 1346 | 104 | | | 520 | 42 | | |
| 5 | 1044 | 106 | | | 634 | 62 | | |

*Obr.32 Počet vrcholů při různé délce kroku u mapy 1*

Při porovnání dosažených výsledků při vyhledávání cesty v jednodušší mapě s různou délkou kroku lze konstatovat, že délka kroku má význam na celkovém počtu nalezených vrcholů. Čím je délka kroku menší, tím je počet nalezených vrcholů vyšší a naopak.

Na obr.33 je porovnání nalezených vrcholů při zadané různé délce kroku u složitější mapy.

| Celkový počet vrcholů a počet vrcholů při nalezení nejkratší cesty při různé délce kroku | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Číslo pokusu | Složitější mapa s délkou kroku 15 | | | | Složitější mapa s délkou kroku 30 | | | |
| | Počet vrcholů | | Průměrný počet | | Počet vrcholů | | Průměrný počet | |
| | Celkem | Nejkratší cesta | Celkem | Nejkratší cesta | Celkem | Nejkratší cesta | Celkem | Nejkratší cesta |
| 1 | 683 | 105 | 404 | 94 | 212 | 48 | 252,6 | 47 |
| 2 | 282 | 94 | | | 141 | 46 | | |
| 3 | 276 | 98 | | | 321 | 47 | | |
| 4 | 324 | 94 | | | 271 | 50 | | |
| 5 | 455 | 79 | | | 318 | 44 | | |

*Obr.33 Počet vrcholů při různé délce kroku u mapy 2*

Při porovnání dosažených výsledků při vyhledávání cesty v jednodušší a složitější mapě je překvapivě nižší počet nalezených vrcholů u složitější mapy. Je to způsobeno tím, že volný prostor, kde by se mohl algoritmus rozrůstat je menší než u jednodušší mapy. Algoritmus má totiž tendenci se rozrůstat na ještě neprohledaná místa.

# 10. ZÁVĚR

Cílem této práce bylo vytvořit ucelený přehled algoritmů používaných pro navigaci mobilních robotů. Navigace mobilních robotů se skládá z lokalizace, mapování a plánování trajektorie. V teoretické části jsou popsány jednotlivé metody lokalizace a metody plánování trajektorie. Poté je vybrána vhodná metoda pro plánování trajektorie ve známé mapě. Jako nejvhodnější algoritmus byl zvolen RRT algoritmus, protože dosahuje velmi dobrých výsledků při hledání cesty.

Dalším cílem bylo implementovat tento algoritmus plánování ve známé mapě. Jako programovací jazyk byl zvolen Delphi 2005. Jedním z výsledků této práce je aplikace, která názorně ukazuje princip a vlastnosti RRT algoritmu. U algoritmu je možné měnit jednotlivé parametry jako například počáteční a cílovou pozici, délku kroku nebo vzdálenost od cíle, která je již považována za cíl. Celá aplikace byla poté zapouzdřena do dynamické knihovny. Dynamická knihovna byla zvolena proto, aby bylo dosaženo univerzálního komunikačního rozhraní.

Ověření funkčnosti vytvořené dynamické knihovny bylo uděláno v programovacím jazyce C a v prostředí LabView. Toto ověření je popsáno v kapitole 8.5 a bylo ukázáno, že dynamická knihovna je schopna komunikovat i v jiném prostředí, než ve kterém byla navržena.

# SEZNAM POUŽITÉ LITERATURY


[1] Wikipedia The Free Encyklopedia, Dostupný z: <http://www.wikipedia.org/>

[2] Košnar K.,: *Mobilní robotika* [pdf dokument], Katedra kybernetiky, Fakulta elektrotechnická, ČVUT v Praze

[3] Topič P.,: *Provedení ověřovacích experimentů lokalizace mobilního robotu*, Bakalářská práce, ÚAI FSI BRNO,2007

[4] Šeda M.,: *Teorie grafů* [pdf dokument], FSI VUT BRNO, 2003

[5] LaValle S. M.,: *Planning Algorithms* [pdf dokument], Cambridge University Press, 2006

[6] Šíma M.,: *Plánování cesty robota ve spojitém prostředí*, Diplomová práce, ÚAI FSI BRNO, 2006

[7] Věchet S., Krejsa J.,: *Mobile robots localization and path planning*, FSI VUT v Brně

[8] Robotika, Dostupný z:http: <//www.robotika.cz/>

[9] Orgoň V.,: *Návrh realizace metody globální lokalizace ve známém prostředí pro mobilní robot*, Diplomová práce ÚAI FSI BRNO, 2007

[10] Šolc, F., Žalud, L.,: *Robotika* [pdf dokument], VUT Brno,2002

[11] Věchet S., Krejsa J.,: Laboratorní kód

[12] Novotný M.,: *Tvorba lokální mapy pracovního prostředí mobilního robotu*, Bakalářská práce ÚMTMB FSI VUT BRNO, 2006

[13] The RRT page, Dostupný z:http: <http://msl.cs.uiuc.edu/rrt/>

# SEZNAM PŘÍLOH

CD-ROM:

- Elektronická podoba této práce – planovani_robotu.pdf
- RRT_algoritmus – program Project1.exe
- Dynamicka_knihovna – program ProjectDLL.dll
- Program_C – program test.exe
- LabView – program Project1.exe